Математическое описание задачи поиска логических закономерностей

1 Выбор методов анализа

В настоящее время наиболее удобным и эффективным способом хранения и доступа к большим объемам информации является организация реляционных баз данных. Современный уровень развития аппаратных и программных средств сделал возможным повсеместное ведение баз данных. Это позволило накопить огромные массивы материалов в различных областях человеческой деятельности. Средства СУБД облегчают пользователям обработку собранного материала, предоставляя возможность осуществлять стандартные операции поиска, ввода и корректировки данных.

Фиксированная структура баз данных гарантирует однозначность схемы формализации вносимой информации в каждую базу данных, что является необходимым условием для возможности проведения более сложной автоматизированной обработки данных, в том числе и методами Data Mining. Вместе с тем, для большинства существующих баз данных их специфика существенно сужает круг методов, которые реально могут быть применены для их анализа. Разнородность составляющих записи полей, а именно: присутствие в записях значений разных типов – числовых, логических, текстовых (часто с преобладанием текстовых) затрудняет или делает невозможным применение к ним методов статистического разведочного анализа данных, ориентированных на обнаружение зависимостей в виде уравнений. Большой объем анализируемых данных и большое количество вариантов значений для каждого поля являются причиной неэффективности использования методов кросс-табуляции.

Методы поиска логических закономерностей в данных были разработаны для разрешения таких ситуаций. Возможность их применения не ограничивается ни типом анализируемых значений, ни объемом анализируемой базы данных. Закономерности, обнаруживаемые в результате применения данных методов, формулируются на простом и понятном человеку языке логических высказываний, что облегчает их интерпретируемость [1]. Поэтому именно поиск логических закономерностей является наиболее возможным, полезным и результативным подходом в анализе реляционных баз данных.

2 Формальная модель исходных данных

Анализируемые данные представляют собой набор кортежей, являющихся выборкой значений определенного набора атрибутов отношений реляционной базы данных. Один из атрибутов данного набора является первичным ключом. Можно считать что каждому значению первичного ключа в выборке соответствует некоторый объект, описываемых множеством свойств – атрибутов и их значений. Обозначим $T = \{t\}$ – исходный набор, в котором каждый объект $t = \{\tau_i\}$ описывается как множество свойств τ_i. $\Theta \equiv \{\omega : \exists t \in T : \exists \tau_i \in t : \tau_i = \omega\}$ – множество всех возможных свойств, элементы которого составляют любое множество свойств (т. е. объект) $e = \{e_i \in \Theta\} \subset \Theta$.

В таком представлении множество является совокупностью нумерованных элементов $t = \{\tau_1, \tau_2, ..., \tau_{\#t}\}$ ($\#t$ – число элементов t). Это обобщенное представление множеств, в котором выполняемые на них отношения и операции являются абстракциями. Введем четыре типа для таких множеств: сочетание, набор, последовательность, строка. Тип множества выбирается в зависимости от специфики конкретной решаемой задачи и определяет отношение и операции, выполняемые на его множествах: вхождение множеств друг в друга, разность множеств, расширение одного множества элементами другого.

Сочетанием (или классическим множеством) будем считать совокупность различных элементов, сортированных в соответствии с некоторым одним и тем же для всех рассматриваемых множеств (например, алфавитным) порядком. Для него справедливы все стандартные отношения и операции, определенные в классической теории множеств. Кроме того, сочетание $t1$, входящее в сочетание $t2$, может быть расширено путем добавления в $t1$ любого элемента из $t2$, не присутствующего в $t1$. Такой вид множеств используется при решении задач поиска закономерностей в базах данных, в которых для описываемых объектов не важен порядок следования значений свойств и все свойства одного объекта различны. Примером такой задачи является задача поиска характерных наборов покупок, совершаемых посетителями универсама за одно посещение, используемая во многих публикациях [2, 3, 4] в качестве иллюстрации применения на практике методов Data Mining. Объектами в базе данных являются покупательские корзины, свойствами которых являются наименования купленных товаров. При этом учитывается только какие именно предметы приобретались без учета их количества и последовательности приобретения.

Набором назовем совокупность, которая может содержать несколько одинаковых элементов, сортированных в соответствии с некоторым одним и тем же для всех рассматриваемых множеств (например, алфавитным) порядком. Для набора вводятся следующие отношения и операции:

- вхождение набора $t1$ в набор $t2$, означает, что все элементы из $t1$ имеются в $t2$ в том же (или более) количестве что и в $t1$ (например $\{ab\} \subset \{aab\}$, но $\{abb\} \not\subset \{ab\}$)

- разность наборов $t1$ и $t2$ состоит из элементов $t1$, в количестве, в котором они содержатся в $t1$ минус количество, в котором они содержатся в $t2$ (например $\{aab\} \setminus \{ab\} = \{a\}$);

- набор $t1$, входящий в набор $t2$, может быть расширен путем добавления в $t1$ любого из элементов $t2$, количество которых в $t2$ больше количества таких же элементов в $t1$ (например, $\{aab\}$ является одним из вариантов расширения $\{ab\}$ в $\{aabb\}$.

Наборы также используются при решении таких задач, в которых для описываемых объектов не важен порядок следования значений свойств, но в отличие от сочетаний, объекты могут иметь повторяющиеся свойства. В качестве примера можно привести ту же задачу, что и для сочетаний, но с учетом в покупательской корзине количества каждого из приобретенных товаров.

Последовательность – это упорядоченный набор элементов. Для последовательностей:

- вхождение последовательности $t1$ в последовательность $t2$ означает, что $t1$ входит в $t2$ как набор, и все элементы из $t1$ содержатся в $t2$ в том же порядке что и в $t1$ (например $\{a\} \subset \{aab\}$, но $\{ba\} \not\subset \{aab\}$)

- разность последовательностей $t1$ и $t2$ получается последовательным удалением из $t1$ элементов $t2$, начиная от начала обеих последовательностей по следующему алгоритму:

  $\#t1 =$ число элементов $t1$;
  $\#t2 =$ число элементов $t2$;
  $i_t1 = 1$;
  $i_t2 = 1$;

  $while \quad ((i_t1 \leq \#t1)\&\&(i_t2 \leq \#t2)) \quad \{$
  $\quad if \quad (t2_{i_t2} = t1_{i_t1}) \quad i_t2++;$
  $\quad else \quad Difference+ = \{t1_{i_t1}\};$
  $\quad i_t1++;$
  $\}$
  $while \quad (i_t1 <= \#t1) \quad \{Difference+ = \{t1_{i_t1}\}; \quad i_t1++;\}$

  (например $\{ababc\} \setminus \{abca\} = \{abc\}$);

- последовательность $t1$, входящая в последовательность $t2$, может быть расширена путем добавления в конец $t1$ любого из элементов $t2$, находящихся в $t2$ после всех элементов, составляющих подпоследовательность $t1$ в $t2$ (например, $\{abca\}$ является одним из вариантов расширения $\{ab\}$ в $\{abbcab\}$.

Последовательности используются в задачах, в которых для описываемых объектов важен порядок следования значений свойств и свойства могут повторяться. В этом случае в приведенном примере задачи о наборах покупок может исследоваться порядок выбора товаров.

Строка (или связная последовательность) представляет собой неразрывную последовательность. Для упрощения определения для строк отношения вхождения и операций разности и расширения введем отношение равенства строк. Равенство строк $t1$ и $t2$ означает, что для любого номера соответствующие ему элементы строк $t1$ и $t2$ либо равны, либо данный номер превышает длины строк $t1$ и $t2$. Тогда:

- вхождение строки $t1$ в строку $t2$, означает, что в $t2$ может быть выделена неразрывная подпоследовательность элементов, равная $t1$. (например $\{bcde\} \subset \{abcdef\}$, но $\{ac\} \not\subset \{abc\}$)
- разность строк $t1$ и $t2$ составляется из подстроки $t1$, следующей в $t1$ сразу за подстрокой, равной $t2$; (например $\{abcdef\} \setminus \{bcd\} = \{ef\}$, $\{abcef\} \setminus \{bcd\} = \{\}$);
- строка $t1$, входящая в строку $t2$, может быть расширена путем добавления в конец $t1$ элемента $t2$, первого из следующих за элементами, составляющих в $t2$ подстроку, равную $t1$ (например, $\{abb\}$ является расширением $\{ab\}$ в $\{aabb\}$.

Строки используются в задачах, в которых для описываемых объектов важна неразрывная цепь следования значений свойств и свойства могут повторяться. В этом случае в приведенном примере задачи о наборах покупок может исследоваться неразрывная последовательность выбора товаров.

3 Поиск значимых множеств

Поиск значимых множеств – один из основных этапов в решении большинства задач обнаружения логических закономерностей в базах данных. Многие виды логических закономерностей, например, последовательности, регулярные эпизоды, являются частными случаями значимых множеств. Другие закономерности, например, условия, ассоциации, формируются на основе значимых множеств.

3.1 Постановка задачи

Дан набор кортежей, представляющий собой выборку значений определенного набора атрибутов отношений базы данных. Элементы кортежей описывают свойства некоторых объектов, образуя для каждого объекта соответствующее ему множество свойств. Значимым множеством для выборки называется такое множество свойств, для которого отношение количества объектов, обладающих данным множеством свойств, к количеству всех объектов, описываемых данными выборки, больше задаваемого пользователем порога. Необходимо получить набор всех значимых множеств для исходной выборки.

3.2 Формальная модель

В формальном виде постановку задачи поиска значимых множеств можно представить следующим образом. Пусть $T = \{t\}$ – исходный набор, в котором каждый объект $t = \{\tau_i\}$ описывается как множество свойств τ_i. Тогда $\Theta \equiv \{\omega : \exists t \in T : \exists \tau_i \in t : \tau_i = \omega\}$ – множество всех возможных свойств, элементы которого составляют любое множество свойств (т. е. объект)
$e = \{e_i \in \Theta\} \subset \Theta$.

$$s(e,T) = \frac{|\{t \in T : e \subset t\}|}{|\{t \in T\}|} - \text{ поддержка } e \subset \Theta \text{ в } T.$$

Необходимо найти набор всех значимых множеств $Large = \{e \subset \Theta : s(e,T) > minsupport\}$, где $minsupport$ – задаваемый нижний порог поддержки значимого множества.

3.3 Алгоритмы решения задачи

3.3.1 Алгоритм Apriory

Алгоритм Apriory был предложен в работе [3]. Его различные модификации приводятся в [5, 4, 6]. Все они принадлежат к классу алгоритмов ограниченного перебора, ограничение которого базируется на априорной идее: любое подмножество значимого множества должно быть значимым [4]. Данное положение объясняется тем фактом, что величина поддержки для множества не может превышать величину поддержки ни одного из его подмножеств.

Ниже представлен алгоритм решения задачи поиска значимых множеств, являющийся частично модифицированной с целью адаптации для реализации версией алгоритма, представленного в работах [3, 5]. $Large$ – набор значимых множеств, $Frontier$ – граничный набор, $Candidate$ – набор кандидатов в значимые множества, $dbsize$ – число объектов в исходном наборе T, $minsupport$ – нижний порог значения поддержки для искомых значимых множеств. Параметр $count$ для каждого объекта хранит текущее (в ходе текущего прохода алгоритма по исходному набору) количество его вхождений в объекты исходного набора.

$Large = \emptyset$;
$Frontier = \emptyset$;
$Candidate = \emptyset$;
$dbsize = 0$;

$\forall t \in T\{$
$dbsize++;$
for $i = 1$ **to** $\#t\{$
if $\{\tau_i\} \in Candidate \quad Candidate.\{\tau_i\}.count++$
else $\{$
$Candidate+ = \{\tau_i\};$
$Candidate.\{\tau_i\}.count = 1;$
$\}$
$\}$
$\}$
$\forall C \in Candidate \quad$ **if** $C.count/dbsize > minsupport\{$
$Large+ = C;$
$Large.C.count = C.count/dbsize;$
$Frontier+ = C;$
$Frontier.C.count = C.count;$
$\}$
while $(Frontier! = \emptyset)\{$
$Candidate = \emptyset;$
$Candidate_f = \emptyset;$
$\forall t \in T \quad \forall f \in Frontier \quad$ **if** $f \subset t\{$
$Cf = \emptyset;$
$Cf_f = \emptyset;$
Extend f**in** $t;$
$\forall C \in Cf\{$
if $C \in Candidate \quad Candidate.C.count++$
else $\{$
$Candidate+ = C;$
$Candidate.C.count = 1;$
$\}$
$\}$
$\forall C \in Cf_f\{$
if $C \in Candidate_f \quad Candidate_f.C.count++$
else $\{$
$Candidate_f+ = C;$
$Candidate_f.C.count = 1;$
$\}$
$\}$
$\}$
$Frontier = \emptyset;$
$\forall C \in Candidate \quad$ **if** $C.count/dbsize > minsupport\{$
$Large+ = C;$
$Large.C.count = C.count/dbsize;$
$\}$

$\forall C \in Candidate_f$ **if** $C.count/dbsize > minsupport\{$
$Large+ = C;$
$Large.C.count = C.count/dbsize;$
$\}$
$\forall C \in Candidate_f$ **if** $C.count/dbsize > minsupport$
$Frontier+ = C;$
$\}$

Указанная в алгоритме конструкция **Extend** f **in** t является процедурой рекурсивного расширения $f \subset t$ в пределах t. Ее действие состоит в следующем: множество свойств f дополняется элементами из t до f_{ex}, пока $s_{possible}$ – максимально возможная поддержка, и $\bar{s}$ – ожидаемая поддержка для f_{ex}. Данные величины являются оценками для реальной еще не подсчитанной на текущий момент поддержки расширения.

Величина поддержки для расширения не может превышать величину поддержки для расширяемого множества. Поэтому $s(f_{ex}, T) \leq s(f) = Frontier.f.count/dbsize$. То есть в T не может содержаться больше чем $Frontier.f.count$ элементов, содержащих f_{ex}. $Candidate.f.count$ – число просмотренных в текущем проходе элементов T, содержащих f. $Candidate.f_{ex}.count$ – число уже просмотренных в текущем проходе элементов T, содержащих f_ex. Тогда $Candidate.f.count - Candidate.f_{ex}.count$ – число элементов T, точно не содержащих f_{ex}, из $Frontier.f.count$, содержащих f. То есть за весь проход не может быть найдено более чем $Frontier.f.count - (Candidate.f.count - Candidate.f_{ex}.count)$ элементов T, содержащих f_{ex}. Таким образом максимально возможная поддержка

$$s_{possible} = \frac{Frontier.f.count-(Candidate.f.count-Candidate.f_{ex}.count)}{dbsize} > > minsupport.$$

Множества f_{ex}, для которых $s_{possible} \leq minsupport$ отбрасываются и дальнейшему расширению не подлежат.

Величина $\bar{s}$, используемая в процедуре расширения множеств свойств, – ожидаемая поддержка для расширения множества свойств. Она вычисляется с учетом известных поддержек расширяющих элементов и возможной поддержки расширения за счет оставшейся не просмотренной в текущем проходе части исходного набора объектов.

$$\bar{s} = (\prod_{i:(\tau_i \in f_{ex})\&\&(\tau_i \notin f)} Large.\tau.count) * * \frac{Frontier.f.count-Candidate.f.count}{dbsize} > > minsupport.$$

Сравнение ожидаемой поддержки с параметром $minsupport$ вводится с целью сокращения числа тех рассматриваемых и расширяемых в текущем проходе множеств свойств, расширения для которых могут не оказаться значимыми множествами. Все, впервые полученные для текущего прохода, f_{ex}

– расширения множества свойств f, для которых $\bar{s} \leq minsupport$, помещаются в Cf_f и будут кандидатами на помещение в граничный набор $Frontier$. Граничный набор $Frontier$ формируется в конце текущего прохода из таких объектов Cf_f, которые являются значимыми множествами, для расширения их в следующем проходе. Остальные расширения, такие для которых $\bar{s} > minsupport$ (которые, как ожидается, окажутся значимыми множествами), помещаются в C_f и продолжают рекурсивно расширяться.

3.3.2 Алгоритм Prefix-Span

Для обоснования возможности применения данного алгоритма необходимо заметить, что в представлении каждого элемента t исходного набора T составляющие его свойства τ_i следуют в некотором фиксированном порядке. То есть он может рассматриваться как последовательность $t = \{\tau_1, \tau_2, ..., \tau_{\#t}\}$. Тогда $e \subset \Theta$ может рассматриваться как префикс в $t \in T$, если $e \subset t$ как подпоследовательность. Постфиксом для префикса $e \subset \Theta$ в $t \in T$ будет являться $\_t = \{\tau_{k+1}, \tau_{k+2}, ..., \tau_{\#t}\}$, где k – индекс последнего элемента e в t.

Алгоритм Prefix-Span, основан на построении дерева решений путем рекурсивного выделения из наборов множеств свойств поднаборов их постфиксов в соответствии со значениями префиксов. На первом этапе работы алгоритма из исходного набора T выбирается набор одноэлементных префиксов $Elements = \{e \in \Theta : s(\{e\}, T) > minsupport\}$. Далее для каждого префикса $e \in Elements$ из элементов исходного набора $t \in T$ строится набор постфиксов $Projected_e$. Процедура выделения постфиксов рекурсивно применяется к каждому из уже построенных наборов постфиксов. В ходе рекурсии образуется дерево решений. Пути движения от верхних уровней к нижним определяют префиксы, соответствующие построенным наборам постфиксов. Движение вглубь дерева идет до тех пор, пока поддержка для соответствующего пути префикса больше заданной $minsupport$. Получаемые таким образом префиксы являются искомыми значимыми множествами.

Ниже представлен алгоритм, основанный на идеях представленного в [7] алгоритма Prefix-Span для поиска регулярных эпизодов, и адаптированный для решения более общей задачи – поиска значимых множеств.

$Large = \emptyset$;
$dbsize = 0$;
$Elements = \emptyset$;
$\forall t \in T\{$
    $dbsize++$;
    for $i = 1$ **to** $\#t\{$
        if $\{\tau_i\} \in Elements$ $Elements.\{\tau_i\}.count++$
        else {
            $Elements+ = \{\tau_i\}$;
            $Elements.\{\tau_i\}.count = 1$;
            }
        }
    }

```
∀e ∈ Elements   if e.count/dbsize > minsupport{
    Large+ = e;
    Large.e.count = e.count/dbsize;
    }
Elements = Large;
∀e ∈ Elements{
    Projected_e = ∅;
    ∀t ∈ T : e ∈ t{
        _t = t.Postfix(e);
        if _t ∈ Projected_e   Projected_e._t.count + +;
        else {
            Projected_e+ = _t;
            Projected_e._t.count = 1;
            }
        }
    CreateNextProjected(e);
    Drop(Projected_e);
    }
Procedure CreateNextProjected(LastPrefix){
    ∀e ∈ Elements{
        Projected_LastPrefix+e = ∅;
        (LastPrefix + e).count = 0;
        ∀t ∈ Projected_LastPrefix : e ∈ t{
            _t = t.Postfix(e);
            if #_t ≤ 0   (LastPrefix + e).count + +;
            if _t ∈ Projected_LastPrefix+e
              Projected_LastPrefix+e._t.count + +;
            else {
                Projected_LastPrefix+e+ = _t;
                Projected_LastPrefix+e._t.count =
                  Projected_LastPrefix.e.count;
                }
            }
        if (LastPrefix + e).count/dbsize > minsupport{
            Large+ = (LastPrefix + e);
            Large.(LastPrefix + e).count =
              (LastPrefix + e).count/dbsize;
            if Projected_LastPrefix+e ≠ ∅
              CreateNextProjected(LastPrefix + e);
            }
        Drop(Projected_LastPrefix+e);
        }
    }
```

3.4 Анализ сложности алгоритмов

Алгоритмы Apriory и Prefix-Span работают по двум различным принципам. Apriory делает несколько проходов по исходному набору T одного и того же объема, имея дело с множествами свойств t фиксированной длины. При этом в каждом проходе строится набор кандидатов в значимые множества, число элементов которого к концу каждого прохода в зависимости от заданного параметра $minsupport$ может стать достаточно большим, причем зависимость обратно пропорциональная. Кроме того с каждым проходом увеличивается длина элементов, составляющих набор кандидатов. При $minsupport = 0$ алгоритм выполняется за один проход, но набор кандидатов при этом будет содержать все возможные подмножества элементов исходного набора T. Большой объем набора кандидатов значительно замедляет работу алгоритма, так как на каждой итерации прохода алгоритма происходит многократное обращение к его элементам, что предполагает выполнение поиска в наборе кандидатов.

Prefix-Span при построении дерева решений на каждом рекурсивном шаге разбивает рассматриваемые части исходного набора на значительно менее объемные поднаборы – наборы постфиксов, длина которых по сравнению с длиной образующих их элементов поднабора, рассматриваемого на текущем уровне рекурсии, меньше как минимум на 1. На первом уровне рекурсии в исходном наборе T делается столько проходов, сколько имеется значимых множеств длины 1. При значении $minsupport = 0$ число проходов равно числу всех возможных элементов $\omega \in \Theta$, составляющих множества свойств исходного набора. Однако на каждом из следующих уровней рекурсии проходы делаются только для построенных на предыдущем уровне наборов постфиксов, длина и число элементов которых значительно сокращается на каждом уровне рекурсии.

Сложность выполнения данных алгоритмов в общем случае зависит от следующих величин: $\#T$ – число элементов исходного набора, $\#t$ – длина элементов $t \in T$, $\#\Theta$ – число возможных элементов, составляющих все $t \in T$, величина поддержки $minsupport$; а также от типа множества, соответствующего элементам T. Для случая множеств-сочетаний, состоящих из различных элементов, при $\#T = n$, $\#t = const = l$, $minsupport = 0$, когда будут рассмотрены все подмножества элементов исходного набора, оценки без учета сложности поиска можно построить следующим образом:
Apriory (в данном случае 1 проход): $n * 2^l$, где 2^l – число всех подмножеств каждого элемента $t \in T$;
Prefix-Span: $n + \sum_{\omega \in \Theta} \#\{t \in T : \omega \in t\} + ... \approx n + n + ... = l * n$. В данном случае преобразование второго и последующих слагаемых в n связано с тем, что на первом уровне рекурсии для каждого элемента $\omega \in \Theta$ отбирается соответствующая ему часть набора элементов T, суммарный объем данных частей будет приблизительно равен n. Аналогичное преобразование выполняется и для слагаемых, соответствующих следующим уровням рекурсии. Всего $l - 1$ таких уровней.

Таким образом, даже для данного простого случая и без учета сложности поиска оценка по длине элементов T для Prefix-Span получается ли-

нейная, а для Apriory – степенная. По $\#T$ они имеют линейную оценку. При увеличении значения $minsupport$ в Apriory вместе с уменьшением числа рассматриваемых в текущем проходе подмножеств элементов T будет увеличиваться число проходов, а для Prefix-Span будет только уменьшаться глубина рекурсии. В случае изменения $minsupport$ достаточно трудно дать аналитическую оценку сложности алгоритмов, однако эксперименты для тестовой выборки одного и того же объема с различным значением $minsupport$ подтверждают утверждение авторов алгоритма Prefix-Span в [7] о том, что данный алгоритм является более эффективным, чем Apriory. Для экспериментов была взята выборка значений трех различных характеристик 1194 топонимов: язык, район, структурная формула. Число уникальных элементов, составляющих кодовую таблицу, соответствующую множеству Θ, равно 1015. Алгоритмы выполнялись с применением системы DMiner, в которой для обоих алгоритмов используются одинаковые структуры данных, что устраняет возможность различия эффективности их реализации. Результаты данных сравнительных экспериментов по времени выполнения алгоритмов системой DMiner на Windows-платформе с процессором Intel Pentium II (330 MHz) следующие:

minsupport	время Apriory	время Prefix-Span
0	11 мин 33 с	9 мин 47 с
0.001	52 с	36 с
0.005	51 с	34 с
0.01	43 с	33 с

4 Генерация правил

Правила – один из основных видов представления логических закономерностей. В зависимости от специфики решаемой задачи правила могут принадлежать к одному из типов: ассоциация (объект, обладающий набором свойств, составляющих посылку правила, также обладает набором свойств, составляющих его следствие), условие (если для объекта выполнены свойства, заданные в посылке, то будут выполнены свойства, определенные в следствии), классификация (при наличии набора свойств посылки объект принадлежит группе, фигурирующей в следствии). При этом тип правила определяется только его семантическим смыслом, а структура и способы построения правил являются одинаковыми для всех типов.

4.1 Постановка задачи

Дан набор кортежей, представляющий собой выборку значений определенного набора атрибутов отношений базы данных. Элементы кортежей описывают свойства некоторых объектов, образуя для каждого объекта соответствующее ему множество свойств. Правилом, выражающим связи между свойствами, называется импликация вида:

$\{Antecedent => Consequent \quad | \quad c, s\}$, где $Antecedent$ и $Consequent$ – множества свойств, выражающие посылку и следствие правила, – фактор уверенности правила, s – степень поддержки правила. Будем говорить, что правило $\{Antecedent => Consequent \quad | \quad c, s\}$ выполнено для исходной выборки с фактором уверенности $0 \leq c \leq 1$, если по крайней мере c доля всех объектов, обладающих свойствами посылки, также обладают свойствами, определенными в следствии. Степень поддержки правила определяется как отношение количества объектов, обладающих свойствами посылки и следствия правила, к количеству всех объектов, описываемых данными исходной выборки.

Необходимо получить набор правил, выражающих связи между элементами исходной выборки и удовлетворяющих заданным значениям уверенности и поддержки.

4.2 Формальная модель

В формальном виде постановку задачи поиска правил можно представить следующим образом. $T = \{t\}$ – исходный набор, в котором каждый объект $t = \{\tau_i\}$ описывается как множество свойств τ_i. $\Theta \equiv \{\omega : \exists t \in T : \exists \tau_i \in t : \tau_i = \omega\}$ – множество всех возможных свойств, элементы которого составляют любое множество свойств (т.е. объект) $e = \{e_i \in \Theta\} \subset \Theta$. $Rule = \{Antecedent => Consequent \quad | \quad c, s\}$ – правило, где $Antecedent \subset \Theta$ и $Consequent \subset \Theta$,

$$s = s(Rule, T) = \frac{|\{t \in T : Antecedent \subset t \quad \&\& \quad Consequent \subset t\}|}{|\{t \in T\}|}$$

– поддержка правила $Rule$ в T,

$$c = c(Rule, T) = \frac{|\{t \in T : Antecedent \subset t \quad \&\& \quad Consequent \subset t\}|}{|\{t \in T : Antecedent \subset t\}|}$$

– степень уверенности правила $Rule$ в T.

Необходимо найти набор всех правил $Rule$, таких что $s > minsupport$ и $c > minconf$, где $minsupport$ и $minconf$ – задаваемые нижние пороги поддержки и степени уверенности правила.

4.3 Алгоритм решения задачи

Генерация правил осуществляется непосредственно из набора значимых множеств. Пусть $Large$ – найденный набор значимых множеств для исходного набора T и нижнего порога поддержки $\leq minsupport$.

Так как каждое подмножество значимого множества также является значимым множеством, а в $Large$ присутствуют все значимые множества, то любое правило можно получить из пары вложенных друг в друга имеющихся в $Large$ множеств. Посылкой будет меньшее по мощности множество, а следствием – их разность (или наоборот, т.к. разность тоже является значимым множеством и содержится в $Large$).

Степенью поддержки правила будет степень поддержки большего по мощности множества, а степенью уверенности – отношение поддержек правила и посылки. Эти величины были получены в процессе нахождения значимых множеств. Таким образом, получаем алгоритм генерации правил:

$Rules = \emptyset$;
$\forall L \in Large | Large.L.count > minsupport$
$\forall K \in Large | Large.K.count < Large.L.count / minconf$
if $K \subset L\{$
$Rule.s = Large.L.count$;
$Rule.c = Large.L.count / Large.K.count$;
$Rule.Antecedent = K$;
$Rule.Consequent = L \setminus K$;
$Rules+ = Rule$;
$\}$

Список литературы

[1] Дюк В., Самойленко А., Data Mining: учебный курс. СПб: Питер, 2001.

[2] Heikki Mannila, Data mining: machine learning, statistics, and databases, SSDBM 1996: 2-9, http://www.cs.helsinki.fi/ mannila/postscripts/ssdbm.ps.

[3] Rakesh Agrawal, Tomasz Imielinski, Arun Swami, Mining Association Rules between Sets of Items in Large Databases // Proc. of the 1993 ACM SIGMOD Conf. Washington DC, USA, (May 1993) pp.207-216.

[4] Rakesh Agrawal, R. Shrikant, Fast Algorythms for Mining Association Rules // Proceedings of the 20th Conference on Very Large Databases. Santiago, Chile, September 1994, pp.487-499.

[5] Jiawei Han, Shojiro Nishio, Hiroyuki Kawano, Wei Wang, Generalization-bazed data mining in object-oriented darabases using an object cube model // Data & Knowledge Engineering, 25 (1998) pp.55-97.

[6] H. Manilla, H.Toivonen, A.I.Verkamo, Discovering frequent episodes in sequences, Proceedings of The First International Conference of Knowledge Discovery and Data Mining (KDD-95), 1995, pp.210-215.

[7] Jian Pei, Juawei Han and others, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, In Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, April 2001, pp.215-224.