УДК 336.74:004.738.5

ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ: ВЫДЕЛЕНИЕ АССОЦИАЦИЙ С ПОМОЩЬЮ АЛГОРИТМА АПРИОРИ

А.Э. Городок
Магистрант кафедры МиИОЭС УО ГрГУ.им. Я.Купалы

Н.В. Марковская
Доцент кафедры МиИОЭС УО ГрГУ.им. Я.Купалы

УО Гродненский государственный университет имени Янки Купалы
E-mail: alexander.gorodok.ed@gmail.com, n.markovskaya@grsu.by

А. Э. Городок
Окончил Гродненский государственный университет имени Янки Купалы. Магистрант кафедры математического и информационного обеспечения экономических систем УО ГрГУ им. Я. Купалы.

Н.В. Марковская
Доцент кафедры математического и информационного обеспечения экономических систем УО ГрГУ им.Я.Купалы, кандидат физико-математических наук, доцент

Аннотация. Современные базы данных имеют очень большие размеры, достигающие гига- и терабайтов, и тенденцию к дальнейшему увеличению. И поэтому, для нахождения ассоциативных правил требуются эффективные масштабируемые алгоритмы, позволяющие решить задачу за приемлемое время. Об одном из таких алгоритмов и пойдет речь.

Ключевые слова: категории, хеш-функции, извлечение правил, ассоциативные правила, алгоритм Apriori.

История

Алгоритм Apriori был предложен в 1994 г. Rakesh Agrawal и Ramakrishnan Srikant, исследовательская группа Almaden IBM) и в настоящее время является одним из наиболее популярных алгоритмов поиска ассоциативных правил. [1]

Свойства алгоритма.

Алгоритм позволяет сократить пространства поиска благодаря свойству антимонотонности множеств, которое утверждает, что если предметный набор Z не является частым, то добавление к нему некоторого нового предмета A не делает его таковым. Иными словами, если Z не является частым, то и Z+A также не будет им.

Начало работы с алгоритмом.

Для того, чтобы было возможно применить алгоритм, необходимо провести предобработку данных: во-первых, привести все данные к бинарному виду; во-вторых, изменить структуру данных.

Пример нормализации:

Номер транзакции	Наименование элемента	Количество
1001	A	2
1001	D	3
1001	E	1
1002	A	2
1002	F	1
1003	B	2
1003	A	2
1003	C	2
...	...	...

TID	A	B	C	D	E	F	G	H	I	K	...
1001	1	0	0	1	1	0	0	0	0	0	...
1002	1	0	0	0	0	1	0	0	0	0	...
1003	1	1	1	0	0	0	0	0	1	0	...

Рисунок *1*. – Таблицы данных до и после нормализации

Свойство антимонотонности.

Выявление часто встречающихся наборов элементов – операция, требующая много вычислительных ресурсов и, соответственно, времени. Примитивный подход к решению данной задачи – простой перебор всех возможных наборов элементов. Это потребует O(2|I|) операций, где |I| – количество элементов. Apriori использует одно из свойств поддержки, гласящее: поддержка любого набора элементов не может превышать минимальной поддержки любого из его подмножеств. Например, поддержка 3-элементного набора {Хлеб, Масло, Молоко} будет всегда меньше или равна поддержке 2-элементных наборов {Хлеб, Масло}, {Хлеб, Молоко}, {Масло, Молоко}. Дело в том, что любая транзакция, содержащая {Хлеб, Масло, Молоко}, также должна содержать {Хлеб, Масло}, {Хлеб, Молоко}, {Масло, Молоко}, причем обратное не верно.

Это свойство носит название анти-монотонности и служит для снижения размерности пространства поиска. Не имей мы в наличии такого свойства, нахождение многоэлементных наборов было бы практически невыполнимой задачей в связи с экспоненциальным ростом вычислений.

Свойству анти-монотонности можно дать и другую формулировку: с ростом размера набора элементов поддержка уменьшается, либо остается такой же. Из всего вышесказанного следует, что любой k-элементный набор будет часто встречающимся тогда и только тогда, когда все его (k-1)-элементные подмножества будут часто встречающимися.

Все возможные наборы элементов из I можно представить в виде решетки, начинающейся с пустого множества, затем на 1 уровне 1-элементные наборы, на 2-м – 2-элементные и т.д. На k уровне представлены k-элементные наборы, связанные со всеми своими (k-1)-элементными подмножествами.

Шаги алгоритма.

На первом шаге алгоритма подсчитываются 1-элементные часто встречающиеся наборы. Для этого необходимо пройтись по всему набору данных и подсчитать для них поддержку, т.е. сколько раз встречается в базе.

Следующие шаги будут состоять из двух частей: генерации потенциально часто встречающихся наборов элементов (их называют кандидатами) и подсчета поддержки для кандидатов.

Описанный выше алгоритм можно записать в виде следующего псевдокода:

```
F1 = {часто встречающиеся 1-элементные наборы}
для (k=2; Fk-1 <> ∅; k++) {
Ck = Apriorigen(Fk-1) // генерация кандидатов
для всех транзакций t ϵ T {
Ct = subset(Ck, t) // удаление избыточных правил
для всех кандидатов c ϵ Ct
c.count ++
}
Fk = { c ϵ Ck | c.count >= minsupport} // отбор кандидатов
}
Результат ∪ Fk
```

Описание алгоритма.

Опишем функцию генерации кандидатов. На это раз нет никакой необходимости вновь обращаться к базе данных. Для того, чтобы получить k-элементные наборы, воспользуемся (k-1)-элементными наборами, которые были определены на предыдущем шаге и являются часто встречающимися.

Вспомним, что наш исходный набор хранится в упорядоченном виде. Генерация кандидатов также будет состоять из двух шагов.

Объединение.

Каждый кандидат Ck будет формироваться путем расширения часто встречающегося набора размера (k-1) добавлением элемента из другого (k-1)- элементного набора.

Приведем алгоритм этой функции Apriorigen в виде небольшого SQL-подобного запроса.

```
insert into Ck
select p.item1, p.item2, …, p.itemk-1, q.itemk-1
From Fk-1 p, Fk-1 q
where p.item1= q.item1, p.item2 = q.item2, … , p.itemk-2 = q.itemk-2, p.itemk-1 < q.itemk-1
```

Удаление избыточных правил.

На основании свойства анти-монотонности, следует удалить все наборы c ϵ Ck если хотя бы одно из его (k-1) подмножеств не является часто встречающимся.

После генерации кандидатов следующей задачей является подсчет поддержки для каждого кандидата. Очевидно, что количество кандидатов может быть очень большим и нужен эффективный способ подсчета. Самый тривиальный способ – сравнить каждую транзакцию с каждым кандидатом. Но это далеко не лучшее решение. Гораздо быстрее и эффективнее использовать подход, основанный на хранении кандидатов в хэш-дереве. Внутренние узлы дерева содержат хэш-таблицы с указателями на потомков, а листья – на кандидатов. Это дерево нам пригодится для быстрого подсчета поддержки для кандидатов.

Хэш-дерево строится каждый раз, когда формируются кандидаты. Первоначально дерево состоит только из корня, который является листом, и не содержит никаких кандидатов-наборов. Каждый раз, когда формируется новый кандидат, он заносится в корень дерева и так до тех пор, пока количество кандидатов в корне-листе не превысит некоего порога. Как только количество кандидатов становится больше порога, корень преобразуется в хэш-таблицу, т.е. становится внутренним узлом, и для него создаются потомки-листья. И все примеры распределяются по узлам-потомкам согласно хэш-значениям элементов, входящих в набор, и т.д. Каждый новый кандидат хэшируется на внутренних узлах, пока он не достигнет первого узла-листа, где он и будет храниться, пока количество наборов опять же не превысит порога.

Хэш-дерево с кандидатами-наборами построено, теперь, используя хэш-дерево, легко подсчитать поддержку для каждого кандидата. Для этого нужно "пропустить" каждую транзакцию через дерево и увеличить счетчики для тех кандидатов, чьи элементы также содержатся и в транзакции, т.е. $Ck \cap Ti = Ck$. На корневом уровне хэш-функция применяется к каждому элементу из транзакции. Далее, на втором уровне, хэш-функция применяется ко вторым элементам и т.д. На k-уровне хэшируется k-элемент. И так до тех пор, пока не достигнем листа. Если кандидат, хранящийся в листе, является подмножеством рассматриваемой транзакции, тогда увеличиваем счетчик поддержки этого кандидата на единицу.

После того, как каждая транзакция из исходного набора данных "пропущена" через дерево, можно проверить удовлетворяют ли значения поддержки кандидатов минимальному порогу. Кандидаты, для которых это условие выполняется, переносятся в разряд часто встречающихся. Кроме того, следует запомнить и поддержку набора, она нам пригодится при извлечении правил. Эти же действия применяются для нахождения (k+1)-элементных наборов и т.д.

После того как найдены все часто встречающиеся наборы элементов, можно приступить непосредственно к генерации правил.

Извлечение правил – менее трудоемкая задача. Во-первых, для подсчета достоверности правила достаточно знать поддержку самого набора и множества, лежащего в условии правила. Например, имеется часто встречающийся набор {A, B, C} и требуется подсчитать достоверность для правила $AB \Rightarrow C$. Поддержка самого набора нам известна, но и его множество {A, B}, лежащее в условии правила, также является часто встречающимся в силу свойства анти-монотонности, и значит его поддержка нам известна. Тогда мы легко сможем подсчитать достоверность. Это избавляет нас от нежелательного просмотра базы транзакций, который потребовался в том случае если бы это поддержка была неизвестна.

Чтобы извлечь правило из часто встречающегося набора F, следует найти все его непустые подмножества. И для каждого подмножества s мы сможем сформулировать правило $s \Rightarrow (F - s)$, если достоверность правила $conf(s \Rightarrow (F - s)) = supp(F)/supp(s)$ не меньше порога minconf.

Заметим, что числитель остается постоянным. Тогда достоверность имеет минимальное значение, если знаменатель имеет максимальное значение, а это происходит в том случае, когда в условии правила имеется набор, состоящий из одного элемента. Все супермножества данного множества имеют меньшую или равную поддержку и, соответственно, большее значение достоверности. Это свойство может быть использовано

при извлечении правил. Если мы начнем извлекать правила, рассматривая сначала только один элемент в условии правила, и это правило имеет необходимую поддержку, тогда все правила, где в условии стоят супермножества этого элемента, также имеют значение достоверности выше заданного порога. Например, если правило A ⇒ BCDE удовлетворяет минимальному порогу достоверности minconf, тогда AB ⇒ CDE также удовлетворяет. Для того, чтобы извлечь все правила используется рекурсивная процедура. Важное замечание: любое правило, составленное из часто встречающегося набора, должно содержать все элементы набора. Например, если набор состоит из элементов {A, B, C}, то правило A ⇒ B не должно рассматриваться. [2]

Пример использования.

Пакет “arules”. Один из самых популярных и хорошо изученных подходов, используемых в анализе данных для выявления интересных связей в обширных базах данных, частые наборы (frequent itemsets) и ассоциативные правила (association rules).

В языке R для использования алгоритма необходим пакет “arules”. Пакет “arules” системы R представляет основу для создания и преобразования входных данных: обеспечивает фундамент для представления, преобразования и анализа транзакционных данных и моделей - частых наборов и ассоциативных правил, - так же обеспечивает интерфейс для реализации в C основанных на идее ассоциативных правил алгоритмов Apriori и Eclat. Эти алгоритмы могут быть использованы для формирования частых наборов, максимальных частых наборов (maximal frequent itemsets), объемлющих частых наборов (closed frequent itemsets) и ассоциативных правил.

Структура данных. Обычные частые наборы или типичные транзакции содержат относительно маленькое количество бинарных признаков по сравнению с общим количеством доступных бинарных признаков, потому естественным представлением такого типа данных является формат разряженных матриц - класс ngCMatrix пакета

Matrix, в котором ngCMatrix - сжатая, разряженная, логическая, столбцовоориентированная матрица, содержащая вектор элементов с значением TRUE в первой строке и указатели на первые элементы столбцов во второй. Нумерация в обоих векторах начинается с 0.

Т.к. более удобно работать с строчно-ориентированными матрицами, в пакете реализован класс itemMatrix - строчно-ориентированный вариант ngCMatrix. В этом случае ngCMatrix может быть доступна методом as(), приводящим itemMatrix к ngCMatrix.

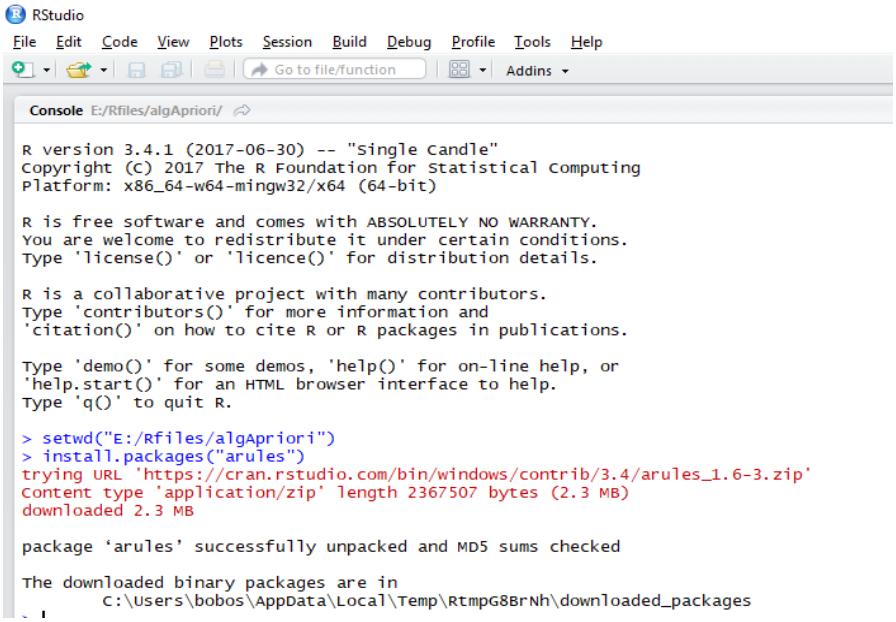

Рисунок *2*. – Установка пакета “arules”

Для визуализации результатов использоваться будет пакет, созданный спецально для названного выше: “arulesViz”.

Большинство методов визуализации описаны Bruzzese and Davino (2008), однако мы добавили больше оттенков, изменение порядка и интерактивные функции (см. Hahsler, 2017). Много визуализации методы принимают дополнительные параметры в качестве списка параметров управления. Хотя мы старались сохранить параметры управления согласованы, доступные параметры управления отличаются незначительно) от визуализации метод к визуализации метод. Полный список параметров со значениями по умолчанию можно получить используя подробный режим.

Несколько интерактивных графиков возвращают набор выбранных правил / наборов элементов. Другие графики могут возвращать другие структуры данных. Например, основанные на графике графики возвращают график (невидимо). Движок "htmlwidget" всегда возвращает объект класса htmlwidget.

Простая визуализация правил ассоциации заключается в использовании точечной диаграммы с двумя интересными мерами на осях. Такое представление можно найти уже в ранней статье Баярдо-младшего и Агравала (1999), когда они обсуждают оптимальные правила. Методом по умолчанию для plot () для правил ассоциации в arulesViz является точечный график с использованием поддержки и достоверности на осях.

```
> install.packages("arulesViz")

  There is a binary version available but the source version is later:
          binary source needs_compilation
arulesViz  1.3-2  1.3-3             FALSE

installing the source package 'arulesViz'

trying URL 'https://cran.rstudio.com/src/contrib/arulesViz_1.3-3.tar.gz'
Content type 'application/x-gzip' length 783780 bytes (765 KB)
downloaded 765 KB

* installing *source* package 'arulesViz' ...
** package 'arulesViz' successfully unpacked and MD5 sums checked
** R
** inst
** preparing package for lazy loading
Warning: package 'arules' was built under R version 3.4.4
** help
*** installing help indices
** building package indices
** installing vignettes
** testing if installed package can be loaded
*** arch - i386
Warning: package 'arules' was built under R version 3.4.4
*** arch - x64
Warning: package 'arules' was built under R version 3.4.4
* DONE (arulesViz)

The downloaded source packages are in
        'C:\Users\bobos\AppData\Local\Temp\RtmpG8BrNh\downloaded_packages'
>
```

Рисунок *3*. – Установка пакета “arulesViz”

Пример создания транзакций для датасета:

```
patterns = random.patterns(nItems = 1000)
> summary(patterns)
set of 2000 itemsets
most frequent items:
item641 item908 item771 item215 item225 (Other)
    80      74      73      55      50    7599

element (itemset/transaction) length distribution:sizes
  1   2   3   4   5   6   7   8   9  10  11  12
114 304 446 465 311 187  94  48  21   8   1   1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   3.000   4.000   3.966   5.000  12.000
summary of quality measures:

    pWeights          pCorrupts
 Min.   :1.160e-07   Min.   :0.0000
 1st Qu.:1.469e-04   1st Qu.:0.2925
 Median :3.293e-04   Median :0.4922
 Mean   :5.000e-04   Mean   :0.4998
 3rd Qu.:6.953e-04   3rd Qu.:0.7066
```

Max. :4.954e-03 Max. :1.0000

includes transaction ID lists: FALSE

> trans = random.transactions(nItems = 1000, nTrans = 1000, method = "agrawal", patterns = patterns)

> image(trans)

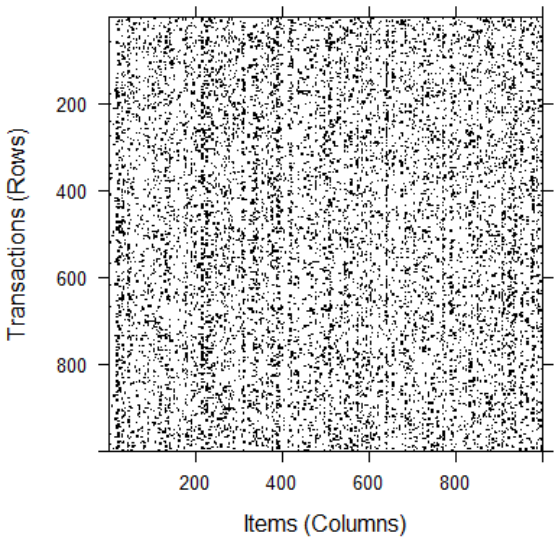

Рисунок *4*. – График сгенерированных транзакций

Пример использования алгоритма.

```
> # implement Apriori Algorithm
> # create transactions
> txn <- as(as.list(as.matrix(grocery)),"transactions")
> # generate rules
> rules = apriori(data = txn, parameter = list(support = 0.01, confidence = 0.5))
```

Особенности применения алгоритма.

Требует ли этот метод обучения или он самообучающийся? Apriori обычно рассматривается как самообучающийся алгоритм, поэтому его часто применяют для поиска интересных шаблонов и отношений. Существует модификация алгоритма Apriori, способная проводить классификацию маркированных данных. В процессе работы алгоритм может быть довольно ресурсоёмким; вычисления могут занять достаточно много времени. Где он используется? Существует огромное количество реализаций Apriori. Одни из самых популярных – это ARtool, Weka и Orange.

Пример работы с датасетом данных людей с корабля «Титаник».

Загрузка датасета и информация о нём.

```
** installing vignettes
** testing if installed package can be loaded
*** arch - i386
Warning: package 'arules' was built under R version 3.4.4
*** arch - x64
Warning: package 'arules' was built under R version 3.4.4
* DONE (arulesViz)

The downloaded source packages are in
        'C:\Users\bobos\AppData\Local\Temp\RtmpG8BrNh\downloaded_packages'
> load("E:/Rfiles/algApriori/titanic.raw.rdata")
> image(titanic.raw)
Error in image.default(titanic.raw) : 'z' must be a matrix
> summary(titanic.raw)
  Class         Sex          Age       Survived
 1st :325   Female: 470   Adult:2092   No :1490
 2nd :285   Male  :1731   Child: 109   Yes: 711
 3rd :706
 Crew:885
>
```

Рисунок *4*. – Детали о наборе данных

Получение правил и информация о них:
Применение алгоритма с стандартными настройками.
Получен набор из 27 правил.

```
> rules <- apriori(titanic.raw)
Error in apriori(titanic.raw) : could not find function "apriori"
> library(arules)
Loading required package: Matrix

Attaching package: 'arules'

The following objects are masked from 'package:base':

    abbreviate, write

Warning message:
package 'arules' was built under R version 3.4.4
> library(arulesViz)
Loading required package: grid
> rules <- apriori(titanic.raw)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 220

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 2201 transaction(s)] done [0.00s].
sorting and recoding items ... [9 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [27 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Рисунок *5*. –Детали о правилах

Результаты анализа правил, приведение их в графиках.

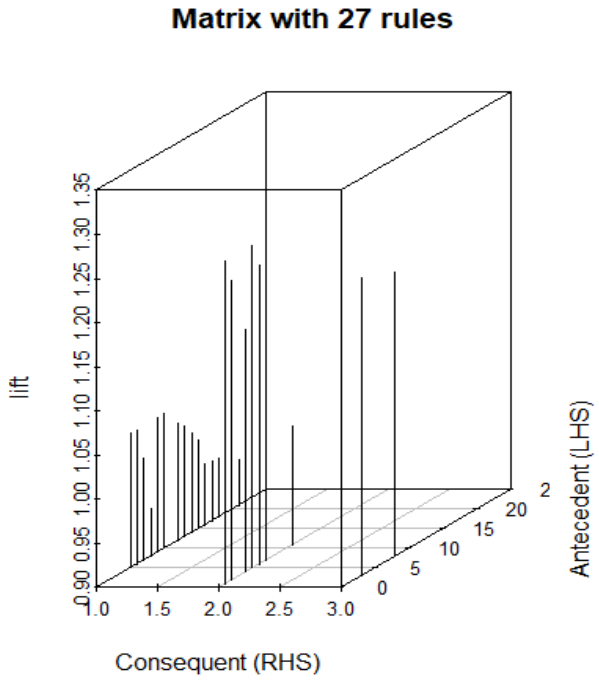

Рисунок *6*. – 3Д-график результатов, показывающая самую высокую концентрацию правил с последствием 2.

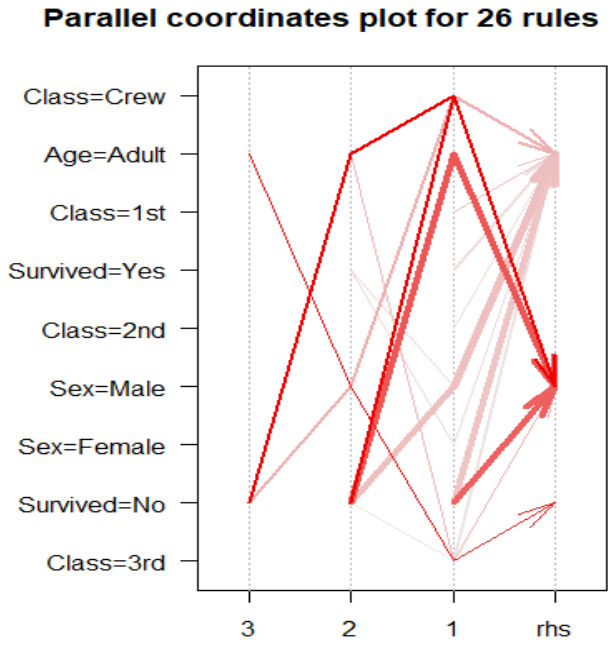

Рисунок 7. – 2Д график результатов, показывающий самую сильную связанность мужского пола с возрастом и классом и с невозможностью выжить.

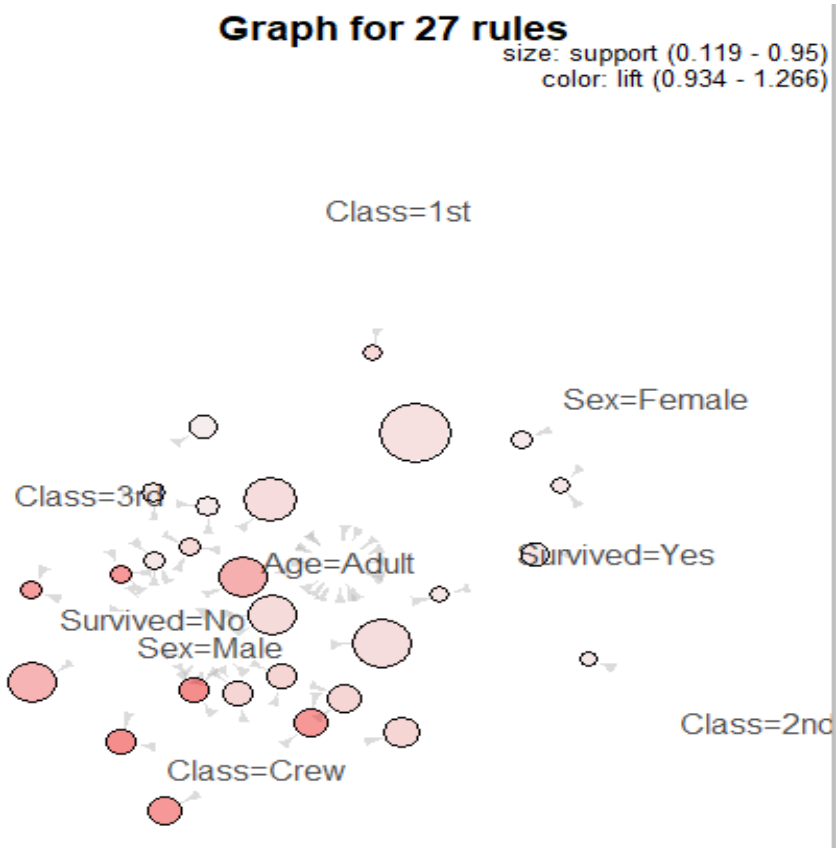

Рисунок *8.* – 2Д график группирующий результаты для анализа

	lhs		rhs	support	confidence	lift	count
[1]	{}	=>	{Age=Adult}	0.9504771	0.9504771	1.0000000	2092
[2]	{Class=2nd}	=>	{Age=Adult}	0.1185825	0.9157895	0.9635051	261
[3]	{Class=1st}	=>	{Age=Adult}	0.1449341	0.9815385	1.0326798	319
[4]	{Sex=Female}	=>	{Age=Adult}	0.1930940	0.9042553	0.9513700	425
[5]	{Class=3rd}	=>	{Age=Adult}	0.2848705	0.8881020	0.9343750	627
[6]	{Survived=Yes}	=>	{Age=Adult}	0.2971377	0.9198312	0.9677574	654
[7]	{Class=Crew}	=>	{Sex=Male}	0.3916402	0.9740113	1.2384742	862
[8]	{Class=Crew}	=>	{Age=Adult}	0.4020900	1.0000000	1.0521033	885
[9]	{Survived=No}	=>	{Sex=Male}	0.6197183	0.9154362	1.1639949	1364
[10]	{Survived=No}	=>	{Age=Adult}	0.6533394	0.9651007	1.0153856	1438
[11]	{Sex=Male}	=>	{Age=Adult}	0.7573830	0.9630272	1.0132040	1667
[12]	{Sex=Female,Survived=Yes}	=>	{Age=Adult}	0.1435711	0.9186047	0.9664669	316
[13]	{Class=3rd,Sex=Male}	=>	{Survived=No}	0.1917310	0.8274510	1.2222950	422
[14]	{Class=3rd,Survived=No}	=>	{Age=Adult}	0.2162653	0.9015152	0.9484870	476
[15]	{Class=3rd,Sex=Male}	=>	{Age=Adult}	0.2099046	0.9058824	0.9530818	462
[16]	{Sex=Male,Survived=Yes}	=>	{Age=Adult}	0.1535666	0.9209809	0.9689670	338
[17]	{Class=Crew,Survived=No}	=>	{Sex=Male}	0.3044071	0.9955423	1.2658514	670
[18]	{Class=Crew,Survived=No}	=>	{Age=Adult}	0.3057701	1.0000000	1.0521033	673
[19]	{Class=Crew,Sex=Male}	=>	{Age=Adult}	0.3916402	1.0000000	1.0521033	862
[20]	{Class=Crew,Age=Adult}	=>	{Sex=Male}	0.3916402	0.9740113	1.2384742	862
[21]	{Sex=Male,Survived=No}	=>	{Age=Adult}	0.6038164	0.9743402	1.0251065	1329
[22]	{Age=Adult,Survived=No}	=>	{Sex=Male}	0.6038164	0.9242003	1.1751385	1329
[23]	{Class=3rd,Sex=Male,Survived=No}	=>	{Age=Adult}	0.1758292	0.9170616	0.9648435	387
[24]	{Class=3rd,Age=Adult,Survived=No}	=>	{Sex=Male}	0.1758292	0.8130252	1.0337773	387
[25]	{Class=3rd,Sex=Male,Age=Adult}	=>	{Survived=No}	0.1758292	0.8376623	1.2373791	387
[26]	{Class=Crew,Sex=Male,Survived=No}	=>	{Age=Adult}	0.3044071	1.0000000	1.0521033	670
[27]	{Class=Crew,Age=Adult,Survived=No}	=>	{Sex=Male}	0.3044071	0.9955423	1.2658514	670

Рисунок *9.* – Список правил.

Заключение

Рассматриваемый алгоритм априори выделяет значимые ассоциации, которые являются ключевыми для датасета и позволяет продолжить работу над ними. За счёт сокращения пространства поиска скорость выше, чем у альтернативных алгоритмов, что особенно важно с современными масштабами данных. В рассматриваемом датасете было выделено 27 правил, самы значимые из которых говорят о плохой выживаемости мужчин первого и второго классов, экипажа. Говорят о хорошей выживаемости девушек. Вероятность выжить повышается как с полом, так и возрастом, чем моложе, тем ближе к выживанию были люди.

Список литературы

[1.] Интернет ресурс - https://en.wikipedia.org/wiki/Apriori_algorithm Режим доступа: 01.07.2019
[2.] Интернет ресурс - https://basegroup.ru/community/articles/apriori Режим доступа 01.07.2019
[3.] Интернет ресурс - http://datascientist.one/apriori-algorithm/ Режим доступа 01.07.2019

INTELLECTUAL DATA ANALYSIS. ALLOCATION OF ASSOCIATIONS WITH THE AID OF THE APRIORI ALGORITHM

A.E. Gorodok
Master student of the Department of Mathematical and Information Support of Economic Systems,
YKSUG

N.V. Markovskaya
Associate professor of the Department of Mathematical and Information Support of Economic Systems,
YKSUG

Yanka Kupala State University of Grodno, Republic of Belarus
E-mail: alexander.gorodok.ed@gmail.com, n.markovskaya@grsu.by

Annotation. Modern databases have very large sizes, reaching gigabytes and terabytes, and a tendency to further increase. Therefore, to find associative rules, effective scalable algorithms are required to solve the problem in a reasonable time. One of these algorithms will be discussed.

Keywords: categories, hash functions, rule extraction, association rules, Apriori algorithm.