Identificación de productos de consumo frecuente con reglas de asociación

Como se señaló en la introducción de este tema, el análisis de la cesta de la compra se utiliza en segundo plano para los sistemas de recomendación que se utilizan en muchos minoristas físicos y en línea. Las reglas de asociación aprendidas indican los artículos que se compran juntos con frecuencia.

El conocimiento de estos patrones proporciona información sobre nuevas formas en las que una cadena de supermercados puede optimizar el inventario, anunciar promociones u organizar la distribución física de la tienda.

Por ejemplo, si los compradores compran con frecuencia café o jugo de naranja con un pastel para el desayuno, es posible que sea posible aumentar las ganancias al reubicar los pasteles más cerca del café y el jugo.

De manera similar, los minoristas en línea pueden usar la información para motores de recomendación dinámicos que sugieran artículos relacionados con los que ya has visto o para realizar un seguimiento después de una visita al sitio web o una compra en línea con un correo electrónico que sugiera artículos adicionales en una práctica llamada **post comercialización activa** (*Active after-marketing*).

En este tutorial, realizaremos un análisis de la cesta de la compra de datos transaccionales de una tienda de comestibles. Al hacerlo, veremos cómo el algoritmo Apriori puede evaluar de manera eficiente un conjunto potencialmente masivo de reglas de asociación.

Las mismas técnicas también se podrían aplicar a muchas otras tareas comerciales, desde recomendaciones de películas hasta sitios de citas o la búsqueda de interacciones peligrosas entre medicamentos.

Paso 1 - recopilación de datos

Nuestro análisis de la cesta de la compra utilizará datos de compra de un mes de funcionamiento en una tienda de comestibles del mundo real. Los datos contienen 9,835 transacciones, o alrededor de 327 transacciones por día (aproximadamente 30 transacciones por hora en un día hábil de 12 horas), lo que sugiere que el minorista no es particularmente grande ni particularmente pequeño.

El conjunto de datos utilizado aquí se adaptó del conjunto de datos Groceries del paquete de R, arules. Para obtener más información, consulta Implications of Probabilistic Data Modeling for Mining Association Rules, Hahsler, M., Hornik, K., Reutterer, T., 2005. En From Data and Information Analysis to Knowledge Engineering, Gaul, W., Vichi, M., Weihs, C., Studies in Classification, Data Analysis, and Knowledge Organization, 2006, pp. 598–605.

Una tienda de comestibles típica ofrece una gran variedad de artículos. Puede haber cinco marcas de leche, una docena de tipos de detergente para ropa y tres marcas de café. Dado el tamaño moderado del minorista en este ejemplo, asumiremos que no está demasiado preocupado por encontrar reglas que se apliquen solo a una marca específica de leche o detergente y, por lo tanto, se eliminan todas las marcas de las compras.

Esto reduce la cantidad de artículos de la tienda a 169 tipos más manejables, utilizando categorías amplias como pollo, comidas congeladas, margarina y refrescos.

Si deseas identificar reglas de asociación muy específicas (por ejemplo, si los clientes prefieren mermelada de uva o de fresa con su mantequilla de maní), necesitarás una enorme cantidad de datos transaccionales. Las grandes cadenas minoristas utilizan bases de datos de muchos millones de transacciones para encontrar asociaciones entre marcas, colores o sabores particulares de los artículos.

¿Tiene alguna idea sobre qué tipos de artículos se pueden comprar juntos? ¿Será el vino y el queso una combinación habitual? ¿El pan y la mantequilla? ¿El té y la miel? Analicemos estos datos y veamos si se pueden confirmar estas suposiciones.

Paso 2 - exploración y preparación de los datos

Los datos transaccionales se almacenan en un formato ligeramente diferente al que hemos utilizado anteriormente. La mayoría de nuestros análisis anteriores utilizaban datos en formato de matriz donde las filas indicaban ejemplos y las columnas indicaban características.

En comparación, los datos transaccionales tienen un formato más libre. Como es habitual, cada fila de los datos especifica un único ejemplo: en este caso, una transacción. Sin embargo, en lugar de tener una cantidad determinada de características, cada registro comprende una lista separada por comas de cualquier cantidad de elementos, desde uno hasta muchos. En esencia, las características pueden diferir de un ejemplo a otro.

Las primeras cinco filas del archivo groceries.csv son las siguientes:

```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
```

Estas líneas indican cinco transacciones independientes en la tienda de comestibles. La primera transacción incluía cuatro artículos: frutas cítricas, pan semiacabado, margarina y sopas preparadas. En comparación, la tercera transacción incluía solo un artículo: leche entera.

Supongamos que intentamos cargar los datos utilizando la función read.csv() como hicimos en los ejemplos (*análisis*) anteriores. R cumpliría gustosamente y leería los datos en un frame de datos en formato de matriz de la siguiente manera:

	V1	V2	V3	V4
1	citrus fruit	semi-finished bread	margarine	ready soups
2	tropical fruit	yogurt	coffee	
3	whole milk			
4	pip fruit	yogurt	cream cheese	meat spreads
5	other vegetables	whole milk	condensed milk	long life bakery product

Figura 1: Leer datos transaccionales en R como un frame de datos causará problemas más adelante.

Notarás que R creó cuatro columnas para almacenar los artículos en los datos transaccionales: V1, V2, V3 y V4. Aunque esto puede parecer razonable, si utilizamos los datos en este formato, nos encontraremos con problemas más adelante. R eligió crear cuatro variables porque la primera línea tenía exactamente cuatro valores separados por comas. Sin embargo, sabemos que las compras de comestibles pueden contener más de cuatro artículos, en el diseño de cuatro columnas, dichas transacciones se dividirán en varias filas de la matriz. Podríamos intentar solucionar esto colocando la transacción con la mayor cantidad de artículos en la parte superior del archivo, pero esto ignora otro asunto (*problema*) más problemático.

Al estructurar los datos de esta manera, R ha construido un conjunto de características que registran no solo los artículos en las transacciones, sino también el orden en que aparecen. Si imaginamos nuestro algoritmo de aprendizaje como un intento de encontrar una relación entre V1, V2, V3 y V4, entonces el artículo de leche entera en V1 podría tratarse de manera diferente que el artículo de leche entera que aparece en V2. En cambio, necesitamos un conjunto de datos que no trate una transacción como un conjunto de posiciones que se deben completar (o no) con artículos específicos, sino como una canasta de mercado que contiene o no contiene cada artículo.

Preparación de datos: creación de una matriz dispersa para datos de transacciones

La solución a este problema utiliza una estructura de datos llamada matriz dispersa. Es posible que recuerdes que utilizamos una matriz dispersa para procesar datos de texto en el tema, Aprendizaje probabilístico: clasificación mediante el método naïve Bayes. Al igual que con el conjunto de datos anterior, cada fila de la matriz dispersa indica una transacción. Sin embargo, la matriz dispersa tiene una columna (es decir, una característica) para cada artículo que podría aparecer en la bolsa de compras de alguien. Dado que hay 169 artículos diferentes en los datos de nuestra tienda de comestibles, nuestra matriz dispersa contendrá 169 columnas.

¿Por qué no almacenar esto simplemente como un frame de datos como hicimos en la mayoría de nuestros análisis anteriores? La razón es que a medida que se agregan transacciones y elementos adicionales, una estructura de datos convencional rápidamente se vuelve demasiado grande para caber en la memoria disponible. Incluso con el conjunto de datos transaccionales relativamente pequeño utilizado aquí, la matriz contiene casi 1.7 millones de celdas, la mayoría de las cuales contienen ceros (de ahí el nombre de matriz "dispersa": hay muy pocos valores distintos de cero).

Dado que no hay ningún beneficio en almacenar todos estos ceros, una matriz dispersa en realidad no almacena la matriz completa en la memoria; solo almacena las celdas que están ocupadas por un elemento. Esto permite que la estructura sea más eficiente en el uso de la memoria que una matriz o frame de datos de tamaño equivalente.

Para crear la estructura de datos de la matriz dispersa a partir de datos transaccionales, podemos utilizar la funcionalidad proporcionada por el paquete arules (reglas de asociación). Instala y carga el paquete utilizando los comandos install.packages("arules") y library(arules).

Para obtener más información sobre el paquete arules, consulta arules - A Computational Environment for Mining Association Rules and Frequent Item Sets, Hahsler, M., Gruen, B., Hornik, K., Journal of Statistical Software, 2005, vol. 14.

Como estamos cargando datos transaccionales, no podemos simplemente usar la función read.csv() utilizada anteriormente. En cambio, arules proporciona una función read.transactions() que es similar a read.csv() con la excepción de que da como resultado una matriz dispersa adecuada para datos transaccionales.

El parámetro sep = "," especifica que los elementos en el archivo de entrada están separados por una coma. Para leer los datos de groceries.csv en una matriz dispersa llamada groceries, escribe la siguiente línea:

```
> groceries <- read.transactions("groceries.csv", sep = ",")
```

Para ver información básica sobre la matriz groceries que acabamos de crear, usa la función summary() en el objeto:

```
> summary(groceries)
```

```
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146
```

El primer bloque de información en la salida proporciona un resumen de la matriz dispersa que creamos.

Las 9835 filas de salida se refieren al número de transacciones, y las 169 columnas indican cada uno de los 169 artículos diferentes que podrían aparecer en la cesta de la compra de alguien. Cada celda de la matriz es un 1 si el artículo se compró para la transacción correspondiente, o un 0 en caso contrario.

El valor de densidad de 0.02609146 (2.6 por ciento) se refiere a la proporción de celdas de la matriz que no son cero.

Dado que hay 9835 * 169 = 1,662,115 posiciones en la matriz, podemos calcular que se compraron un total de 1,662,115 * 0.02609146 = 43,367 artículos durante los 30 días de funcionamiento de la tienda (ignorando el hecho de que podrían haberse comprado duplicados de los mismos artículos).

Con un paso adicional, podemos determinar que la transacción media contenía 43367 / 9835 = 4.409 artículos de alimentación distintos. Por supuesto, si miramos un poco más abajo en la salida, veremos que ya se ha proporcionado la cantidad media de artículos por transacción.

El siguiente bloque de la salida summary() enumera los artículos que se encontraron con mayor frecuencia en los datos transaccionales. Como 2513 / 9835 = 0.2555, podemos determinar que la leche entera apareció en el 25.6 por ciento de las transacciones.

Otras verduras, panecillos, refrescos y yogur completan la lista de otros artículos comunes, como se muestra a continuación:

```
most frequent items:
      whole milk other vegetables       rolls/buns             soda           yogurt          (Other)
            2513             1903             1809             1715             1372            34055
```

También se nos presenta un conjunto de estadísticas sobre el tamaño de las transacciones. Un total de 2,159 transacciones contenían un solo artículo, mientras que una transacción tenía 32 artículos. El primer cuartil y el tamaño de compra medio son dos y tres artículos respectivamente, lo que implica que el 25 por ciento de las transacciones contenían dos o menos artículos y aproximadamente la mitad contenían tres artículos o menos. La media de 4,409 artículos por transacción coincide con el valor que calculamos a mano:

```
element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23
2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46   29   14   14    9   11    4    6
  24   26   27   28   29   32
   1    1    1    1    3    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   4.409   6.000  32.000
```

Por último, la parte inferior del resultado incluye información adicional sobre los metadatos que pueden estar asociados con la matriz de artículos, como jerarquías de artículos o etiquetas. No hemos utilizado estas funciones avanzadas, pero el resultado sigue indicando que los datos tienen etiquetas.

La función read.transactions() agregó estos elementos automáticamente al cargarlos utilizando los nombres de los elementos en el archivo CSV original, y se muestran las primeras tres etiquetas (en orden alfabético):

```
includes extended item information - examples:
            labels
1 abrasive cleaner
2 artif. sweetener
3   baby cosmetics
```

Ten en cuenta que el paquete arules representa los elementos internamente utilizando números de identificación de elementos numéricos sin conexión con el elemento en el mundo real. De manera predeterminada, la mayoría de las funciones arules decodificarán estos números utilizando las etiquetas de los elementos.

Sin embargo, para ilustrar los identificadores numéricos, podemos examinar las dos primeras transacciones sin decodificar utilizando el llamado formato “largo”. En los datos transaccionales de formato largo, cada fila es un elemento único de una sola transacción en lugar de que cada fila represente una sola transacción con varios elementos. Por ejemplo, debido a que la primera y la segunda transacción tenían cuatro y tres elementos, respectivamente, el formato largo representa estas transacciones en siete filas:

> head(toLongFormat(groceries, decode = FALSE), n = 7)

```
  TID item
1   1   30
2   1   89
3   1  119
4   1  133
5   2   34
6   2  158
7   2  168
```

En esta representación de los datos transaccionales, la columna denominada TID hace referencia al identificador de transacción (es decir, la primera o la segunda canasta de mercado) y la columna denominada artículo hace referencia al número de identificación interno asignado al artículo.

Como la primera transacción contenía {frutas cítricas, margarina, sopas preparadas y pan semiacabado}, podemos suponer que el ID de artículo 30 se refiere a frutas cítricas y 89 se refiere a margarina.

El paquete arules, por supuesto, incluye funciones para examinar los datos de transacciones en formatos más intuitivos. Para ver el contenido de la matriz dispersa, utiliza la función inspect() en combinación con los operadores vectoriales de R. Las primeras cinco transacciones se pueden ver de la siguiente manera:

> inspect(groceries[1:5])

```
    items
[1] {citrus fruit, margarine, ready soups, semi-finished bread}
[2] {coffee, tropical fruit, yogurt}
[3] {whole milk}
[4] {cream cheese, meat spreads, pip fruit, yogurt}
[5] {condensed milk, long life bakery product, other vegetables, whole milk}
```

Cuando se formatean con la función inspect(), los datos no se ven muy diferentes de lo que habíamos visto en el archivo CSV original.
Debido a que el objeto de comestibles se almacena como una matriz de artículos dispersa, la notación [fila, columna] se puede utilizar para examinar los artículos deseados, así como las transacciones deseadas.

Al utilizar esto con la función itemFrequency(), podemos ver la proporción de todas las transacciones que contienen el artículo especificado. Por ejemplo, para ver el nivel de compatibilidad de los tres primeros artículos en todas las filas de los datos de comestibles, utiliza el siguiente comando:

> itemFrequency(groceries[, 1:3]

```
abrasive cleaner artif. sweetener   baby cosmetics
     0.0035587189     0.0032536858     0.0006100661
```

Observa que los artículos en la matriz dispersa están organizados en columnas en orden alfabético. Los limpiadores abrasivos y los edulcorantes artificiales se encuentran en aproximadamente el 0.3 por ciento de las transacciones, mientras que los cosméticos para bebés se encuentran en aproximadamente el 0.06 por ciento de las transacciones.

Visualización de la compatibilidad de artículos: gráficos de frecuencia de artículos

Para presentar estas estadísticas visualmente, utiliza la función itemFrequencyPlot(). Esto crea un gráfico de barras que representa la proporción de transacciones que contienen artículos específicos. Dado que los datos transaccionales contienen una gran cantidad de artículos, a menudo necesitarás limitar los que aparecen en el gráfico para producir un gráfico legible.

Si deseas mostrar los artículos que aparecen en una proporción mínima de transacciones, utilice itemFrequencyPlot() con el parámetro de compatibilidad:

```
> itemFrequencyPlot(groceries, support = 0.1)
```

Como se muestra en el siguiente gráfico, esto da como resultado un histograma que muestra los ocho artículos en los datos de comestibles con al menos un 10 por ciento de apoyo:

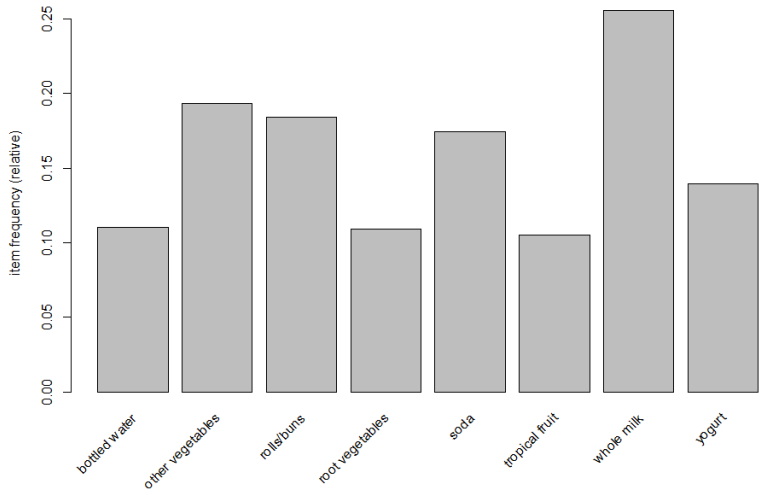

Figura 2: Niveles de apoyo para todos los artículos comestibles en al menos el 10 por ciento de las transacciones.

Si prefieres limitar el gráfico a una cantidad específica de artículos, utiliza la función con el parámetro topN:

```
> itemFrequencyPlot(groceries, topN = 20)
```

El histograma se ordena entonces por apoyo decreciente, como se muestra en el siguiente diagrama para los 20 artículos principales en los datos de comestibles:

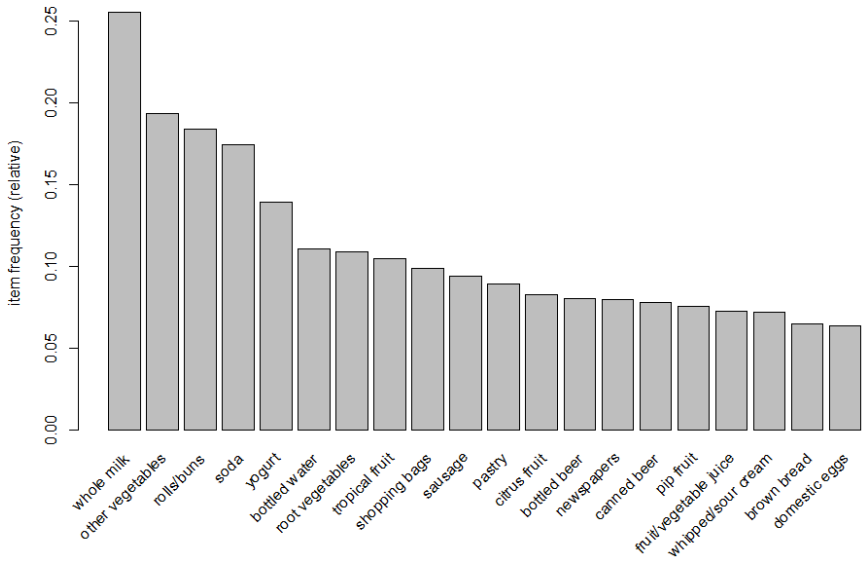

Figura 3: Niveles de apoyo para los 20 artículos principales de comestibles.

Visualización de los datos de transacciones: trazado de la matriz dispersa

Además de observar artículos específicos, también es posible obtener una vista aérea de toda la matriz dispersa utilizando la función image(). Por supuesto, debido a que la matriz en sí es muy grande, generalmente es mejor solicitar un subconjunto de toda la matriz. El comando para visualizar la matriz dispersa para las primeras cinco transacciones es el siguiente:

```
> image(groceries[1:5])
```

El diagrama resultante muestra una matriz con 5 filas y 169 columnas, que indican las 5 transacciones y los 169 elementos posibles que solicitamos. Las celdas de la matriz están rellenas de negro para las transacciones (filas) en las que se compró el elemento (columna).

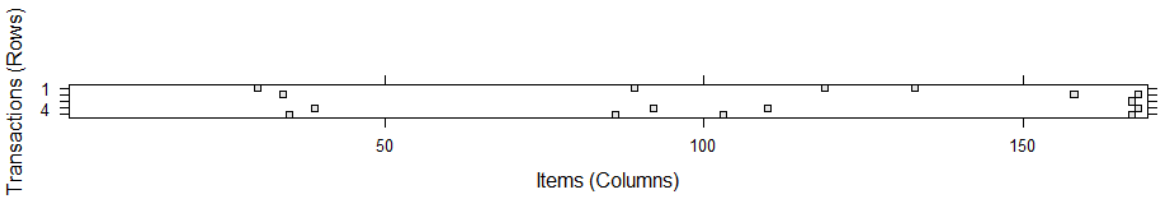

Figura 4: Una visualización de la matriz dispersa para las primeras cinco transacciones.

Aunque la figura 4 es pequeña y puede resultar un poco difícil de leer, puedes ver que la primera, cuarta y quinta transacciones contenían cuatro elementos cada una, ya que sus filas tienen cuatro celdas rellenas. En el lado derecho del diagrama, también puedes ver que las filas tres y cinco, y las filas dos y cuatro, comparten un elemento en común.

Esta visualización puede ser una herramienta útil para explorar datos transaccionales. Por un lado, puede ayudar con la identificación de posibles problemas de datos.

Las columnas que se llenan hasta el final podrían indicar los artículos que se compran en cada transacción, un problema que podría surgir, tal vez, si el nombre o el número de identificación de un minorista se incluyeran inadvertidamente en el conjunto de datos de transacciones.

Además, los patrones en el diagrama pueden ayudar a revelar segmentos interesantes de transacciones y artículos, en particular si los datos están ordenados de maneras interesantes.

Por ejemplo, si las transacciones están ordenadas por fecha, los patrones en los puntos negros podrían revelar efectos estacionales en la cantidad o los tipos de artículos comprados. Tal vez alrededor de Navidad o Hanukkah, los juguetes sean más comunes; alrededor de Halloween, tal vez los dulces se vuelvan populares.

Este tipo de visualización podría ser especialmente eficaz si los artículos también estuvieran ordenados en categorías. En la mayoría de los casos, sin embargo, el gráfico se verá bastante aleatorio, como la estática en una pantalla de televisión.

Ten en cuenta que esta visualización no será tan útil para bases de datos de transacciones extremadamente grandes porque las celdas serán demasiado pequeñas para discernirlas. Aún así, al combinarla con la función sample(), puedes ver la matriz dispersa para un conjunto de transacciones muestreadas aleatoriamente. El comando para crear una selección aleatoria de 100 transacciones es el siguiente:

```
> image(sample(groceries, 100))
```

Esto crea un diagrama de matriz con 100 filas y 169 columnas:

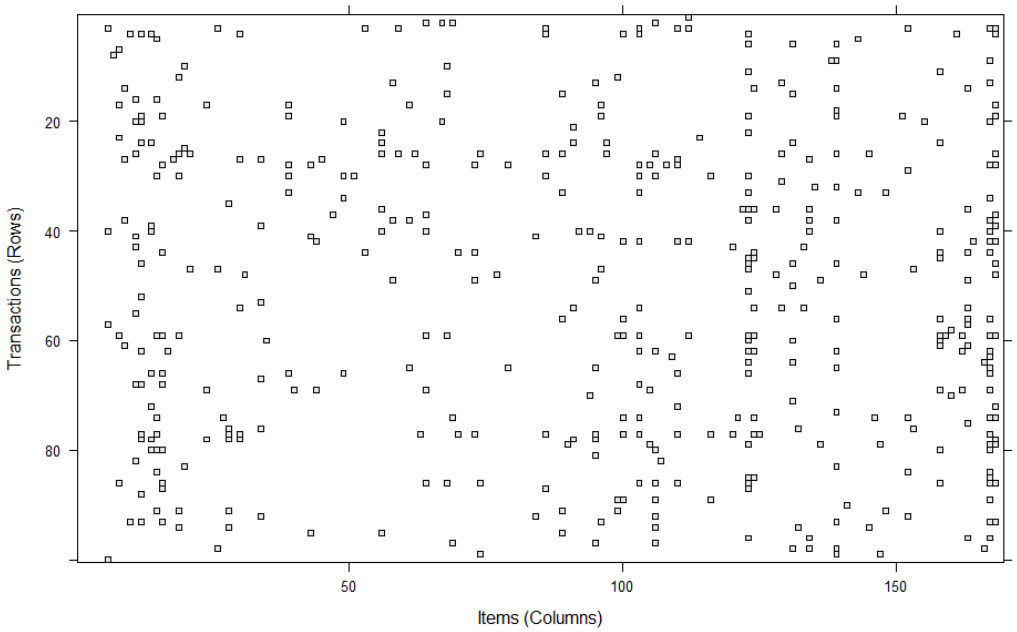

Figura 5: Una visualización de la matriz dispersa para 100 transacciones seleccionadas aleatoriamente.

Unas pocas columnas parecen bastante pobladas, lo que indica algunos artículos muy populares en la tienda.
Sin embargo, la distribución de puntos parece bastante aleatoria en general. Sin nada más digno de mención, continuemos con nuestro análisis.

Paso 3 - entrenamiento de un modelo con los datos

Una vez completada la preparación de los datos, ahora podemos trabajar en la búsqueda de asociaciones entre los artículos del carrito de compras. Usaremos una implementación del algoritmo Apriori en el paquete arules que hemos estado usando para explorar y preparar los datos de comestibles. Deberás instalar y cargar este paquete si aún no lo has hecho.

La siguiente tabla (figura 6) muestra la sintaxis para crear conjuntos de reglas con la función apriori():

Aunque ejecutar la función apriori() es sencillo, a veces puede ser necesario realizar una buena cantidad de pruebas y errores para encontrar los parámetros de soporte y confianza que produzcan una cantidad razonable de reglas de asociación. Si estableces estos niveles demasiado altos, es posible que no encuentres reglas o que encuentres reglas que sean demasiado genéricas para ser muy útiles. Por otro lado, un umbral demasiado bajo puede dar como resultado una cantidad de reglas difícil de manejar. Peor aún, la operación puede llevar mucho tiempo o quedarse sin memoria durante la fase de aprendizaje.

Association rule syntax

using the `apriori()` function in the `arules` package

Finding association rules:

```
myrules <- apriori(data = mydata, parameter =
  list(support = 0.1, confidence = 0.8, minlen = 1))
```

- `data` is a sparse item matrix holding transactional data
- `support` specifies the minimum required rule support
- `confidence` specifies the minimum required rule confidence
- `minlen` specifies the minimum required rule items

The function will return a rules object storing all rules that meet the minimum criteria.

Examining association rules:

```
inspect(myrules)
```

- `myrules` is a set of association rules from the `apriori()` function

This will output the association rules to the screen. Vector operators can be used on `myrules` to choose a specific rule or rules to view.

Example:

```
groceryrules <- apriori(groceries, parameter =
 list(support = 0.01, confidence = 0.25, minlen = 2))
inspect(groceryrules[1:3])
```

Figura 6: Sintaxis de aprendizaje de reglas de asociación Apriori.

En los datos de comestibles, el uso de la configuración predeterminada de soporte = 0.1 y confianza = 0.8 conduce a un resultado decepcionante. Aunque se ha omitido el resultado completo por razones de brevedad, el resultado final es un conjunto de cero reglas:

> apriori(groceries)

```
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 TRUE

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

Absolute minimum support count: 983

set item appearances ...[0 item(s)] done [0.01s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.26s].
sorting and recoding items ... [8 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [0 rule(s)] done [0.00s].
creating S4 object  ... done [0.04s].
set of 0 rules
```

Obviamente, debemos ampliar un poco la búsqueda.

Si lo piensas, este resultado no debería haber sido demasiado sorprendente. Como el soporte = 0.1 por defecto, para generar una regla, un artículo debe haber aparecido en al menos 0.1 * 9,385 = 938.5 transacciones. Como solo ocho artículos aparecieron con esta frecuencia en nuestros datos, no sorprende que no encontráramos ninguna regla.

Una forma de abordar el problema de establecer un soporte mínimo es pensar en la menor cantidad de transacciones necesarias para que una parte interesada considere que un patrón es interesante. Por ejemplo, se podría argumentar que si un artículo se compra dos veces al día (unas 60 veces en un mes de datos), entonces puede ser importante. A partir de ahí, es posible calcular el nivel de soporte necesario para encontrar solo reglas que coincidan con al menos esa cantidad de transacciones. Como 60 de 9,835 equivale aproximadamente a 0.006, intentaremos establecer el soporte allí primero.

Establecer la confianza mínima implica un equilibrio delicado. Por un lado, si el nivel de confianza es demasiado bajo, podemos vernos abrumados por muchas reglas poco fiables, como docenas de reglas que indican artículos que se compran juntos con frecuencia por pura casualidad, como el pan y las pilas. ¿Cómo sabríamos entonces dónde destinar nuestro presupuesto publicitario? Por otro lado, si fijamos el nivel de confianza demasiado alto, estaremos limitados a reglas que son obvias o inevitables, como el hecho de que un detector de humo siempre se compra en combinación con pilas. En este caso, es poco probable que acercar los detectores de humo a las pilas genere ingresos adicionales, ya que los dos artículos ya se compraron casi siempre juntos.

El nivel de confianza mínimo adecuado depende en gran medida de los objetivos de tu análisis. Si comienzas con un valor conservador, siempre puedes reducirlo para ampliar la búsqueda si no encuentras información útil.

Comenzaremos con un umbral de confianza de 0.25, lo que significa que para que se incluya en los resultados, la regla debe ser correcta al menos el 25 por ciento de las veces. Esto eliminará las reglas menos fiables y nos permitirá cierto margen para modificar el comportamiento con promociones específicas.

Ahora estamos listos para generar algunas reglas. Además de los parámetros de confianza y soporte mínimos, es útil establecer minlen = 2 para eliminar las reglas que contienen menos de dos elementos. Esto evita que se creen reglas poco interesantes simplemente porque el artículo se compra con frecuencia, por ejemplo, {} => leche entera. Esta regla cumple con los parámetros de confianza y soporte mínimos porque la leche entera se compra en más del 25 por ciento de las transacciones, pero no es una información muy útil.

El comando completo para buscar un conjunto de reglas de asociación mediante el algoritmo Apriori es el siguiente:

```
> groceryrules <- apriori(groceries, parameter = list(support =
+   0.006, confidence = 0.25, minlen = 2))
```

Las primeras líneas de salida describen las configuraciones de parámetros que especificamos, así como otras que permanecieron configuradas con sus valores predeterminados; para obtener definiciones de estos, utilice el comando de ayuda ?APparameter.

El segundo conjunto de líneas muestra parámetros de control algorítmicos en segundo plano que pueden resultar útiles para conjuntos de datos mucho más grandes, ya que controlan compensaciones computacionales como la optimización de la velocidad o el uso de la memoria. Para obtener información sobre estos parámetros, utiliza el comando de ayuda ¿APcontrol:

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
       0.25    0.1    1 none FALSE            TRUE       5   0.006      2     10  rules TRUE

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

A continuación, la salida incluye información sobre los pasos en la ejecución del algoritmo Apriori:

```
Absolute minimum support count: 59

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [109 item(s)] done [0.00s].
creating transaction tree ... done [0.05s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [463 rule(s)] done [0.00s].
creating S4 object  ... done [0.02s].
```

Dado el pequeño tamaño del conjunto de datos transaccionales, la mayoría de las filas muestran pasos que prácticamente no tomaron tiempo para ejecutarse (indicados como [0.00s] en la salida aquí, pero su salida puede variar levemente según la capacidad de la computadora).

El recuento de *soporte mínimo absoluto* se refiere al recuento más pequeño de transacciones que cumplirían con el umbral de soporte de 0.006 que especificamos. Como 0.006 * 9835 = 59.01, el algoritmo requiere que los artículos aparezcan en un mínimo de 59 transacciones. Los subconjuntos de verificación de tamaño 1 2 3 4 que se obtienen como resultado sugieren que el algoritmo probó conjuntos de i-artículos de 1, 2, 3 y 4 artículos antes de detener el proceso de iteración y escribir el conjunto final de 463 reglas.

El resultado final de la función apriori() es un objeto rules, al que podemos echar un vistazo escribiendo su nombre:

> groceryrules
set of 463 rules

¡Nuestro objeto groceryrules contiene un conjunto bastante grande de reglas de asociación! Para determinar si alguna de ellas es útil, tendremos que investigar más a fondo.

Paso 4 - evaluación del rendimiento del modelo

Para obtener una descripción general de alto nivel de las reglas de asociación, podemos usar summary() de la siguiente manera. La distribución de la longitud de las reglas nos indica cuántas reglas tienen cada recuento de artículos. En nuestro conjunto de reglas, 150 reglas tienen solo dos artículos, mientras que 297 tienen tres y 16 tienen cuatro. Las estadísticas de resumen asociadas con esta distribución también se proporcionan en las primeras líneas de salida:

> summary(groceryrules)

```
set of 463 rules

rule length distribution (lhs + rhs):sizes
  2   3   4
150 297  16

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.000   3.000   2.711   3.000   4.000
```

Como se señaló en la salida anterior, el tamaño de la regla se calcula como el total tanto del lado izquierdo (izq.) como del lado derecho (der.) de la regla. Esto significa que una regla como {pan} => {mantequilla} son dos elementos y {mantequilla de maní, mermelada} => {pan} son tres.

A continuación, vemos las estadísticas de resumen de las medidas de calidad de la regla, que incluyen el soporte y la confianza, así como la cobertura, la elevación y el recuento:

```
summary of quality measures:
    support           confidence        coverage             lift            count
 Min.   :0.006101   Min.   :0.2500   Min.   :0.009964   Min.   :0.9932   Min.   : 60.0
 1st Qu.:0.007117   1st Qu.:0.2971   1st Qu.:0.018709   1st Qu.:1.6229   1st Qu.: 70.0
 Median :0.008744   Median :0.3554   Median :0.024809   Median :1.9332   Median : 86.0
 Mean   :0.011539   Mean   :0.3786   Mean   :0.032608   Mean   :2.0351   Mean   :113.5
 3rd Qu.:0.012303   3rd Qu.:0.4495   3rd Qu.:0.035892   3rd Qu.:2.3565   3rd Qu.:121.0
 Max.   :0.074835   Max.   :0.6600   Max.   :0.255516   Max.   :3.9565   Max.   :736.0
```

Las medidas de soporte y confianza no deberían sorprendernos mucho, ya que las usamos como criterios de selección para las reglas. Podríamos alarmarnos si la mayoría o todas las reglas tuvieran un soporte y una confianza muy cerca de los umbrales mínimos, ya que esto significaría que podríamos haber establecido el listón demasiado alto. Este no es el caso aquí, ya que hay muchas reglas con valores mucho más altos de cada uno.

Las medidas de recuento (*conteo*) y cobertura están estrechamente relacionadas con el soporte y la confianza. Como se define aquí, el **conteo** es simplemente el numerador de la métrica de apoyo o el número (en lugar de la proporción) de transacciones que contenían el elemento. Debido a que el recuento mínimo absoluto de apoyo fue 59, no sorprende que el recuento mínimo observado de 60 esté cerca del parámetro establecido. El recuento máximo de 736 sugiere que un elemento apareció en 736 de 9835 transacciones; esto se relaciona con el apoyo máximo observado como 736 / 9835 = 0.074835.

La **cobertura** de una regla de asociación es simplemente el apoyo del lado izquierdo de la regla, pero tiene una interpretación útil en el mundo real: puede entenderse como la probabilidad de que una regla se aplique a cualquier transacción dada en el conjunto de datos, seleccionada al azar. Por lo tanto, la cobertura mínima de 0.009964 sugiere que la regla menos aplicable cubre solo alrededor del uno por ciento de las transacciones; la cobertura máxima de 0.255516 sugiere que al menos una regla cubre más del 25 por ciento de las transacciones. Seguramente esta regla involucra a la leche entera, ya que es el único producto que aparece en tantas transacciones.

La última columna es una métrica que aún no hemos considerado. El valor de **elevación** (*lift*) de una regla mide la probabilidad de que un artículo o conjunto de artículos se compre en relación con su tasa de compra típica, dado que se sabe que se ha comprado otro artículo o conjunto de artículos. Esto se define mediante la siguiente ecuación:

$$\text{elevacion}(X \rightarrow Y) = \frac{\text{confianza}(X \rightarrow Y)}{soporte(Y)}$$

A diferencia de la confianza, donde el orden de los artículos importa, el valor de elevación (X → Y) es lo mismo que el valor de elevación (Y → X).

Por ejemplo, supongamos que en una tienda de comestibles la mayoría de las personas compran leche y pan. Por pura casualidad, esperaríamos encontrar muchas transacciones tanto de leche como de pan. Sin embargo, si el valor de elevación (leche → pan) es mayor que 1, esto implica que los dos artículos se encuentran juntos con más frecuencia de lo esperado por pura casualidad. En otras palabras, alguien que compra uno de los artículos tiene más probabilidades de comprar el otro. Por lo tanto, un valor de elevación alto es un fuerte indicador de que una regla es importante y refleja una verdadera conexión entre los

artículos, y que la regla será útil para fines comerciales. Sin embargo, ten en cuenta que esto solo es así para conjuntos de datos transaccionales suficientemente grandes; Los valores de elevación pueden exagerarse para los elementos con un soporte bajo.

Un par de autores del paquete apriori han propuesto nuevas métricas llamadas **hyper-lift** e **hyper confidence** para abordar las deficiencias de estas medidas para datos con elementos poco comunes. Para obtener más información, consulta M. Hahsler y K. Hornik, New Probabilistic Interest Measures for Association Rules (2018). https://arxiv.org/pdf/0803.0966.pdf.

En la sección final de la salida summary(), recibimos información de minería, que nos informa sobre cómo se eligieron las reglas. Aquí, vemos que los datos de comestibles, que contenían 9,835 transacciones, se utilizaron para construir reglas con un soporte mínimo de 0.006 y una confianza mínima de 0.25:

```
mining info:
      data ntransactions support confidence
 groceries          9835   0.006       0.25
                                                                                call
 apriori(data = groceries, parameter = list(support = 0.006, confidence = 0.25, minlen = 2))
```

Podemos echar un vistazo a reglas específicas utilizando la función inspect(). Por ejemplo, las primeras tres reglas del objeto GroceryRules pueden verse de la siguiente manera:

> inspect(groceryrules[1:3])

```
    lhs                rhs                support     confidence coverage   lift     count
[1] {potted plants} => {whole milk}       0.006914082 0.4000000  0.01728521 1.565460 68
[2] {pasta}         => {whole milk}       0.006100661 0.4054054  0.01504830 1.586614 60
[3] {herbs}         => {root vegetables}  0.007015760 0.4312500  0.01626843 3.956477 69
```

La primera regla puede leerse en lenguaje sencillo como “si un cliente compra plantas en macetas, también comprará leche entera”. Con un soporte de aproximadamente 0.007 y una confianza de 0.400, podemos determinar que esta regla cubre aproximadamente el 0.7 por ciento de las transacciones y es correcta en el 40 por ciento de las compras que involucran plantas en macetas. El valor de elevación nos indica cuántas más probabilidades hay de que un cliente compre leche entera en relación con el cliente promedio, dado que compró una planta en maceta.

Como sabemos que aproximadamente el 25.6 por ciento de los clientes compraron leche entera (soporte), mientras que el 40 por ciento de los clientes que compraron una planta en maceta compraron leche entera (confianza), podemos calcular la elevación como 0.40 / 0.256 = 1.56, que coincide con el valor que se muestra.

Ten en cuenta que la columna etiquetada como soporte indica el soporte para la regla, no el soporte solo para el lado izquierdo o derecho. La columna etiquetada como cobertura es el soporte para el lado izquierdo.

Aunque la confianza y la elevación (‘impulso’) son altos, ¿{plantas en maceta} → {leche entera} parece una regla muy útil? Probablemente no, ya que no parece haber una razón lógica por la que alguien tenga más probabilidades de comprar leche con una planta en maceta. Sin embargo, nuestros datos sugieren lo contrario. ¿Cómo podemos entender este hecho?

Un enfoque común es tomar las reglas de asociación y dividirlas en las siguientes tres categorías:

- Accionables
- Triviales
- Inexplicables

Obviamente, el objetivo de un análisis de la cesta de la compra es encontrar reglas accionables que proporcionen una perspectiva clara e interesante. Algunas reglas son claras y otras son interesantes; es menos común encontrar una combinación de ambos factores.

Las llamadas reglas **triviales** incluyen todas aquellas reglas que son tan obvias que no vale la pena mencionarlas: son claras, pero no interesantes. Supongamos que tu eres un consultor de marketing al que le pagan grandes sumas de dinero para identificar nuevas oportunidades de promoción cruzada de artículos. Si informas el hallazgo de que {pañales} → {fórmula}, probablemente no te volverán a invitar a otro trabajo de consultoría.

Las reglas triviales también pueden colarse disfrazadas de resultados más interesantes. Por ejemplo, supongamos que encontraste una asociación entre una marca particular de cereales para niños y una película animada popular. Este hallazgo no es muy esclarecedor si el personaje principal de la película está en el frente de la caja de cereales.

Las reglas son **inexplicables** si la conexión entre los artículos es tan poco clara que averiguar cómo usar la información es imposible o casi imposible. La regla puede ser simplemente un patrón aleatorio en los datos, por ejemplo, una regla que indica que {pepinillos} → {helado de chocolate} puede deberse a un solo cliente cuya esposa embarazada tenía antojos regulares de combinaciones extrañas de alimentos.

Las mejores reglas son las joyas ocultas - los conocimientos no descubiertos que solo parecen obvios una vez descubiertos. Si se dispone de tiempo suficiente, se podrían evaluar todas y

cada una de las reglas para encontrar las joyas. Sin embargo, los científicos de datos que trabajan en el análisis pueden no ser los mejores jueces para determinar si una regla es procesable, trivial o inexplicable.
En consecuencia, es probable que surjan mejores reglas mediante la colaboración con los expertos en el dominio responsables de la gestión de la cadena minorista, que pueden ayudar a interpretar los hallazgos. En la siguiente sección, facilitaremos dicho intercambio empleando métodos para ordenar y exportar las reglas aprendidas de modo que los resultados más interesantes aparezcan en la parte superior.

Paso 5 - mejorar el rendimiento del modelo

Los expertos en la materia pueden ser capaces de identificar reglas útiles muy rápidamente, pero sería un mal uso de su tiempo pedirles que evalúen cientos o miles de reglas. Por lo tanto, es útil poder ordenar las reglas según diferentes criterios y obtenerlas de R en un formato que se pueda compartir con los equipos de marketing y examinar con más profundidad.

De esta manera, podemos mejorar el rendimiento de nuestras reglas al hacer que los resultados sean más prácticos.

Si te encuentras con limitaciones de memoria o si Apriori tarda demasiado en ejecutarse, también puede ser posible mejorar el rendimiento computacional del proceso de extracción de reglas de asociación mediante el uso de un algoritmo más reciente.

Ordenar el conjunto de reglas de asociación

Dependiendo de los objetivos del análisis de la cesta de la compra, las reglas más útiles pueden ser aquellas con el mayor soporte, confianza o elevación. El paquete arules funciona con la función sort() de R para permitir la reordenación de la lista de reglas de modo que las que tengan los valores más altos o más bajos de la medida de calidad aparezcan primero.

Para reordenar el objeto de las reglas de supermercado, podemos usar sort() mientras especificamos un valor de “soporte”, “confianza” o “elevación” en el parámetro by. Al combinar la ordenación con operadores vectoriales, podemos obtener una cantidad específica de reglas interesantes. Por ejemplo, las cinco mejores reglas según la estadística de elevación se pueden examinar utilizando el siguiente comando:

```
> inspect(sort(groceryrules, by = "lift")[1:5])
```

El resultado es el siguiente:

```
    lhs                                                rhs                  support     confidence coverage
[1] {herbs}                                         => {root vegetables}    0.007015760 0.4312500  0.01626843
[2] {berries}                                       => {whipped/sour cream} 0.009049314 0.2721713  0.03324860
[3] {other vegetables, tropical fruit, whole milk} => {root vegetables}    0.007015760 0.4107143  0.01708185
[4] {beef, other vegetables}                        => {root vegetables}    0.007930859 0.4020619  0.01972547
[5] {other vegetables, tropical fruit}              => {pip fruit}          0.009456024 0.2634561  0.03589222
    lift     count
[1] 3.956477 69
[2] 3.796886 89
[3] 3.768074 69
[4] 3.688692 78
[5] 3.482649 93
```

Estas reglas parecen ser más interesantes que las que analizamos anteriormente. La primera regla, con un aumento de aproximadamente 3.96, implica que las personas que compran hierbas tienen casi cuatro veces más probabilidades de comprar tubérculos que el cliente típico, tal vez para un guiso de algún tipo.

La segunda regla también es interesante. Es tres veces más probable encontrar crema batida en un carrito de compras con bayas que en otros carritos, lo que sugiere tal vez una combinación de postres.

De manera predeterminada, el orden de clasificación es decreciente, lo que significa que los valores más grandes aparecen primero. Para invertir este orden, agrega un parámetro adicional, decreasing = FALSE.

Tomando subconjuntos de reglas de asociación

Supongamos que, dada la regla anterior, el equipo de marketing está entusiasmado con la posibilidad de crear un anuncio para promocionar las bayas (*berries*), que ahora están en temporada. Sin embargo, antes de finalizar la campaña, te piden que investigues si las bayas se compran a menudo con otros artículos. Para responder a esta pregunta, necesitaremos encontrar todas las reglas que incluyen bayas de alguna forma.

La función subset() proporciona un método para buscar subconjuntos de transacciones, artículos o reglas. Para utilizarla para encontrar cualquier regla que incluya bayas, utiliza el siguiente comando.
Esto almacenará las reglas en un nuevo objeto llamado berryrules:

```
> berryrules <- subset(groceryrules, items %in% "berries")
```

Podemos inspeccionar las reglas como lo habíamos hecho con el conjunto más grande:

```
> inspect(berryrules)
```

El resultado es el siguiente conjunto de reglas:

```
    lhs          rhs                    support     confidence coverage  lift     count
[1] {berries} => {whipped/sour cream} 0.009049314 0.2721713  0.0332486 3.796886  89
[2] {berries} => {yogurt}             0.010574479 0.3180428  0.0332486 2.279848 104
[3] {berries} => {other vegetables}   0.010269446 0.3088685  0.0332486 1.596280 101
[4] {berries} => {whole milk}         0.011794611 0.3547401  0.0332486 1.388328 116
```

Hay cuatro reglas que incluyen bayas, dos de las cuales parecen ser lo suficientemente interesantes como para ser consideradas procesables. Además de la crema batida, las bayas también se compran frecuentemente con yogur, una combinación que podría servir bien para el desayuno o el almuerzo, así como para el postre.

La función subset() es muy poderosa. Los criterios para elegir el subconjunto se pueden definir con varias palabras clave y operadores:

- La palabra clave items, explicada anteriormente, coincide con un elemento que aparece en cualquier parte de la regla. Para limitar el subconjunto a donde la coincidencia ocurre solo en el lado izquierdo o derecho, utiliza lhs o rhs en su lugar.
- El operador %in% significa que al menos uno de los elementos debe encontrarse en la lista que definiste. Si deseas que alguna regla coincida con bayas o yogur, puedes escribir items %in% c("berries", "yogurt").
- Hay operadores adicionales disponibles para coincidencia parcial (%pin%) y coincidencia completa (%ain%). La coincidencia parcial te permite encontrar frutas cítricas y tropicales con una sola búsqueda: items %pin% "fruit". La coincidencia completa requiere que todos los elementos de la lista estén presentes. Por ejemplo, items %ain% c("berries", "yogurt") encuentra solo reglas con bayas y yogur.
- Los subconjuntos también pueden limitarse por soporte, confianza o elevación. Por ejemplo, confianza > 0.50 limitaría las reglas a aquellas con confianza mayor al 50 por ciento.
- Los criterios de coincidencia se pueden combinar con operadores lógicos estándar de R como AND (&), OR (|) y NOT (!).

Con estas opciones, puede limitar la selección de reglas para que sean tan específicas o generales como desee.

Guardar las reglas de asociación en un archivo o frame de datos

Para compartir los resultados de tu análisis de la cesta de la compra, puedes guardar las reglas en un archivo CSV con la función write(). Esto generará un archivo CSV que se puede utilizar en la mayoría de los programas de hojas de cálculo, incluido Microsoft Excel:

```
> write(groceryrules, file = "groceryrules.csv",
+   sep = ",", quote = TRUE, row.names = FALSE)
```

A veces también es conveniente convertir las reglas en un frame de datos R. Esto se puede lograr utilizando la función as(), de la siguiente manera:

```
> groceryrules_df <- as(groceryrules, "data.frame")
```

Esto crea un frame de datos con las reglas en formato de caracteres y vectores numéricos para soporte, confianza, cobertura, elevación y recuento:

```
> str(groceryrules_df)
```

```
'data.frame':   463 obs. of  6 variables:
 $ rules     : chr  "{potted plants} => {whole milk}" "{pasta} => {whole milk}" "{herbs} => {root vegetables}" "{herbs} => {other vegetables}" ...
 $ support   : num  0.00691 0.0061 0.00702 0.00773 0.00773 ...
 $ confidence: num  0.4 0.405 0.431 0.475 0.475 ...
 $ coverage  : num  0.0173 0.015 0.0163 0.0163 0.0163 ...
 $ lift      : num  1.57 1.59 3.96 2.45 1.86 ...
 $ count     : int  68 60 69 76 76 69 70 67 63 88 ...
```

Guardar las reglas en un frame de datos puede resultar útil si deseas realizar un procesamiento adicional en las reglas o necesitas exportarlas a otra base de datos.

Uso del algoritmo Eclat para lograr una mayor eficiencia

El algoritmo Eclat, que recibe su nombre por el uso de métodos de “agrupamiento de conjuntos de elementos de clases de equivalencia y recorrido de red (*lattice*) ascendente”, es un algoritmo de aprendizaje de reglas de asociación un poco más moderno y sustancialmente más rápido. Si bien los detalles de implementación están fuera del alcance de este curso, se puede entender como un pariente cercano de Apriori; también asume que todos los subconjuntos de conjuntos de elementos frecuentes también son frecuentes.

Sin embargo, Eclat puede buscar incluso menos subconjuntos utilizando trucos inteligentes que brindan atajos para identificar los conjuntos de elementos frecuentes potencialmente máximos y buscar solo los subconjuntos de estos conjuntos de elementos. Mientras que Apriori es una forma de algoritmo de amplitud porque busca en todo antes de buscar en profundidad, Eclat se considera un algoritmo de profundidad porque se sumerge hasta el punto final y busca solo en todo lo amplio que sea necesario. Para algunos casos de uso, esto puede generar una ganancia de rendimiento de un orden de magnitud y un menor consumo de memoria.

Para obtener más información sobre Eclat, consulta New Algorithms for Fast Discovery of Association Rules, Zaki, M. J., Parthasarathy, S., Ogihara, M., Li, W., KDD 97 Proceedings, 1997.

Una desventaja clave de la búsqueda rápida de Eclat es que omite la fase en Apriori en la que se calcula la confianza. Supone que una vez que se obtienen los conjuntos de elementos con alto soporte, las asociaciones más útiles se pueden identificar más tarde, ya sea manualmente a través de una prueba visual subjetiva o mediante otra ronda de procesamiento para calcular métricas como la confianza y la elevación. Dicho esto, el paquete arules hace que sea tan fácil aplicar Eclat como Apriori, a pesar del paso adicional en el proceso.

Comenzamos con la función eclat() y configuramos el parámetro de soporte en 0.006 como antes; Sin embargo, ten en cuenta que la confianza no se establece en esta etapa:

```
> groceryitemsets_eclat <- eclat(groceries, support = 0.006)
```

Aquí se omite parte de la salida, pero las últimas líneas son similares a las que obtuvimos de la función apriori(), con la excepción clave de que se escribieron 747 conjuntos de elementos en lugar de 463 reglas:

```
Absolute minimum support count: 59

create itemset ...
set transactions ...[169 item(s), 9835 transaction(s)] done [0.05s].
sorting and recoding items ... [109 item(s)] done [0.00s].
creating sparse bit matrix ... [109 row(s), 9835 column(s)] done [0.14s].
writing  ... [747 set(s)] done [0.26s].
Creating S4 object  ... done [0.13s].
```

El objeto de conjunto de elementos de Eclat resultante se puede utilizar con la función inspect() como lo hicimos con el objeto de reglas de Apriori. El siguiente comando muestra los primeros cinco conjuntos de elementos:

```
> inspect(groceryitemsets_eclat[1:5])
```

```
    items                          support     count
[1] {potted plants, whole milk} 0.006914082 68
[2] {pasta, whole milk}         0.006100661 60
[3] {herbs, whole milk}         0.007727504 76
[4] {herbs, other vegetables}   0.007727504 76
[5] {herbs, root vegetables}    0.007015760 69
```

Para generar reglas a partir de los conjuntos de elementos, utiliza la función ruleInduction() con el valor del parámetro de confianza deseado de la siguiente manera:

```
> groceryrules_eclat <- ruleInduction(groceryitemsets_eclat,
+   confidence = 0.25)
```

Con el soporte y la confianza establecidos en los valores anteriores de 0.006 y 0.25, respectivamente, no sorprende que el algoritmo Eclat haya producido el mismo conjunto de 463 reglas que Apriori:

```
> groceryrules_eclat
set of 463 rules
```

El objeto de reglas resultante se puede inspeccionar igual que antes:

```
> inspect(groceryrules_eclat[1:5])
```

```
    lhs                rhs                support     confidence lift
[1] {potted plants} => {whole milk}       0.006914082 0.4000000  1.565460
[2] {pasta}         => {whole milk}       0.006100661 0.4054054  1.586614
[3] {herbs}         => {whole milk}       0.007727504 0.4750000  1.858983
[4] {herbs}         => {other vegetables} 0.007727504 0.4750000  2.454874
[5] {herbs}         => {root vegetables}  0.007015760 0.4312500  3.956477
```

Dada la facilidad de uso con cualquiera de los dos métodos, si tienes un conjunto de datos transaccionales muy grande, puede que valga la pena probar Eclat y Apriori en muestras aleatorias más pequeñas de transacciones para ver si uno supera al otro.