



REGLAS DE ASOCIACIÓN: INTRODUCCIÓN

TRABAJO EN LABORATORIO

Jesús Alcalá Fernández

Research Group: Soft Computing and Intelligent Information Systems

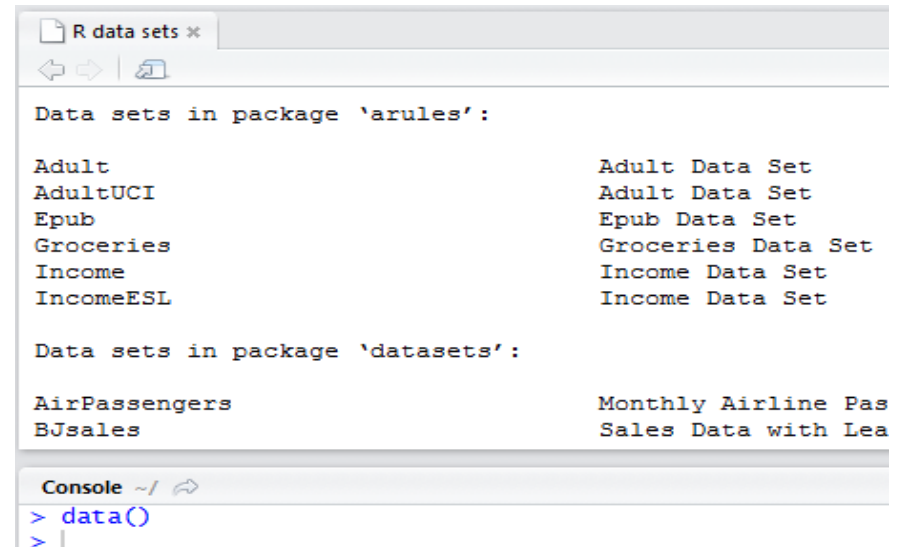
Departamento de Ciencias de la Computación e I.A.

Universidad de Granada



Package “arules”

- Aunque también podemos encontrar en el paquete RWeka de R algún método de extracción de reglas, el paquete “arules” (versión 1.3-1, necesaria al menos la versión 3.2 de R para poder instalarla) es el que proporciona la infraestructura básica para poder extraer y analizar reglas de asociación.
- Para poderlo utilizar debemos primero instalarlo y activarlo:
 - > `install.packages(“arules”)`
 - > `library(arules)`
- A parte de las bases de datos (BD) ya instalados en R, el paquete arules nos proporciona algunos BDs de ejemplo. La función *data* nos informa sobre las BDs previamente instaladas
 - > `data()`



```
R data sets *  
Data sets in package 'arules':  
Adult           Adult Data Set  
AdultUCI        Adult Data Set  
Epub            Epub Data Set  
Groceries       Groceries Data Set  
Income          Income Data Set  
IncomeESL       Income Data Set  
  
Data sets in package 'datasets':  
AirPassengers   Monthly Airline Pas  
BJsales         Sales Data with Lea  
  
Console ~/   
> data()  
> |
```

Preparación de las BDS

- Para poder utilizar una BD con el paquete `arules`, la BD tiene que estar guardada como un conjunto de transacciones.
- Preparación de la BD “AdultUCI” (*BD obtenida desde el repositorio de UCI machine learning, con 48842 instancias y 15 atributos/variables, de los que 6 son atributos numéricos*):

```
> data("AdultUCI")  ## Lo cargamos en nuestra zona de trabajo  
> dim(AdultUCI)     ## Consultamos sus dimensiones  
> AdultUCI[1:2,]    ## Vemos las 2 primeras filas para ver los atributos sus tipos
```

Preparación de las BDS

- De los 6 atributos continuos:
 - 2 los eliminamos porque aportan información redundante: *fnlwgt* y *education-num*
 - Los 4 restantes los dividimos en intervalos:
 - Age: Lo cortamos en los niveles Young (0-25), Middle-aged (26-45), Senior (46-65) y Old (66+).
 - *hours-per-week*: Lo cortamos en los niveles Part-time (0-25), Full-time (25-40), Over-time (40-60) y Too-much (60+).
 - *capital-gain and capital-losseach*: Lo cortamos en los niveles None (0), Low ($0 < \text{mediana de los valores mayores que } 0$) y High ($\geq \text{mediana}$).

Preparación de las BDS

➤ Instrucciones para Eliminar *fnlwgt* y *education-num*

```
> AdultUCI[["fnlwgt"]] = NULL
```

```
> AdultUCI[["education-num"]] = NULL
```

➤ Instrucciones para hacer los cortes:

```
> AdultUCI[["age"]] = ordered( cut ( AdultUCI[["age"]], c(15,25,45,65,100) ) ,
```

```
labels = c ("Young", "Middle-aged", "Senior", "Old"))
```

```
> AdultUCI[["hours-per-week"]] = ordered( cut ( AdultUCI[["hours-per-week"]], c(0,25,40,60,168) ) ,  
labels = c("Part-time", "Full-time", "Over-time", "Workaholic"))
```

```
> AdultUCI[["capital-gain"]] = ordered( cut ( AdultUCI[["capital-gain"]], c(-Inf,0,median(AdultUCI[["  
capital-gain"]][AdultUCI[["capital-gain"]]>0]), Inf) ) , labels = c("None", "Low", "High"))
```

```
> AdultUCI[["capital-loss"]] = ordered( cut ( AdultUCI[["capital-loss"]], c(-Inf,0, median(AdultUCI[["  
capital-loss"]][AdultUCI[["capital-loss"]]>0]), Inf) ) , labels = c("None", "Low", "High"))
```

➤ Convertimos el data.frame en un conjunto de transacciones con la función *as* y lo guardamos en la variable *Adult*:

```
> Adult <- as(AdultUCI, "transactions")
```

```
> Adult
```

Obtener información BDS

➤ Para ver un resumen de la BD hacemos:

> `summary(Adult)`

transactions as itemMatrix in sparse format with
48842 rows (elements/itemsets/transactions) and
115 columns (items) and a density of 0.1089939

most frequent items:

capital-loss=None	capital-gain=None	native-country=United-States	race=White	workclass=Private	(Other)
46560	44807	43832	41762	33906	401333

element (itemset/transaction) length distribution:

sizes

9	10	11	12	1	
19	971	2067	15623	30162	
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
9.00	12.00	13.00	12.53	13.00	13.00

includes extended item information - examples:

	labels	variables	levels
1	age = Young	age	Young
2	age = Middle-aged	age	Middle-aged
3	age = Senior	age	Senior

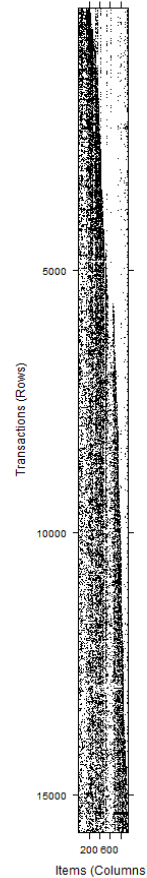
includes extended transaction information - examples:

transaction	ID
1	1
2	2
3	3

Obtener información BDS

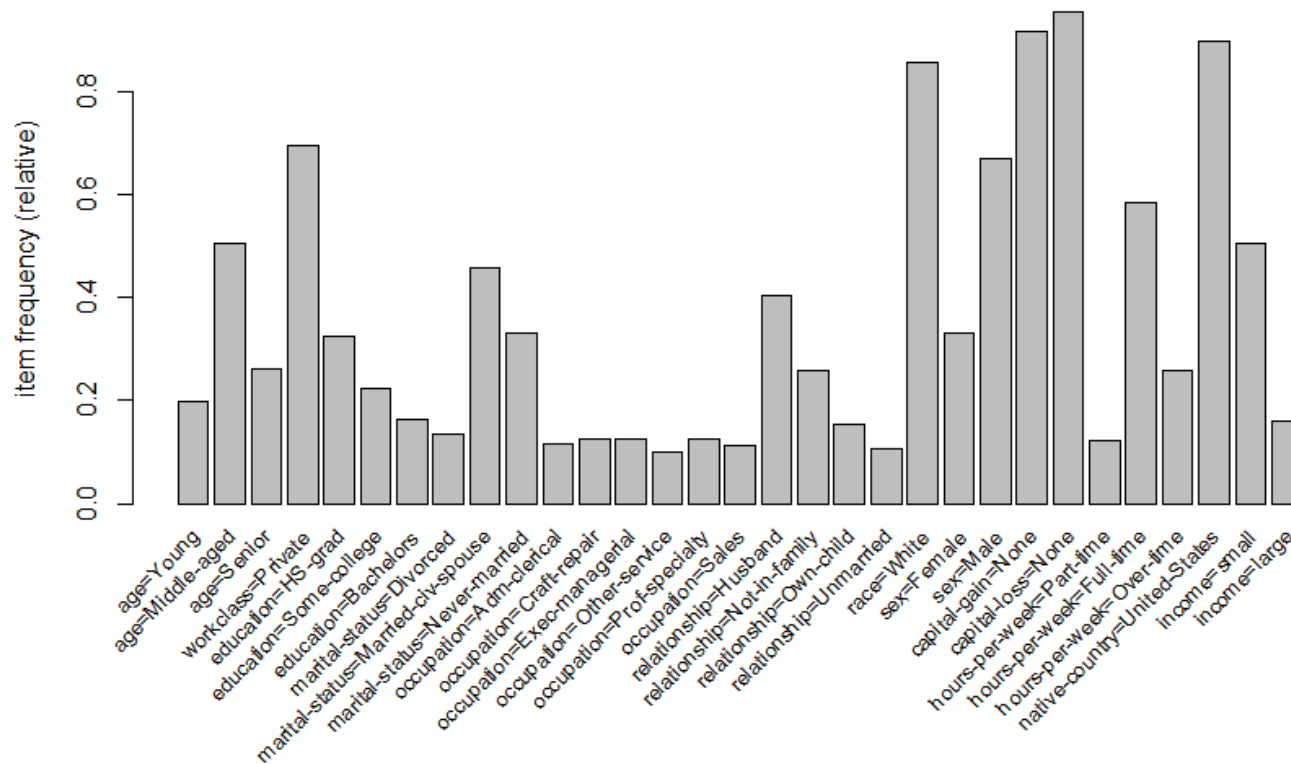
- Representar gráficamente la distribución de los items en las transacciones. Como en Adult cada transacción tienen un valor cada atributo/variable, usamos para probarlo la BD Epub (15729 transacciones y 936 items)

```
> data(Epub)
> summary(Epub)
> image(Epub)
```



Items frecuentes

- Para ver gráficamente que items son los más importantes podemos ejecutar *itemFrequencyPlot*. En este caso usamos mínimo soporte 0.1 y reducimos el tamaño de los títulos con el parámetro *cex.names*:
- > `itemFrequencyPlot(Adult, support = 0.1, cex.names=0.8)`

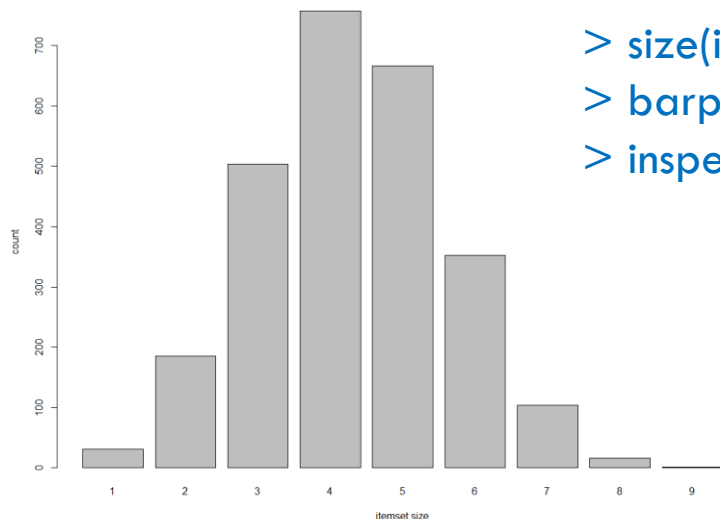


Extraer Itemsets frecuentes

- Usamos apriori para extraer los itemsets frecuentes con mínimo soporte 0.1. Para ello ponemos en la lista de parámetros `target="frequent"` :

```
> iAdult <- apriori(Adult, parameter = list(support = 0.1, target="frequent"))
> iAdult <- sort(iAdult, by="support") # Los ordenamos por el valor del soporte
> inspect(head(iAdult, n=10))          # Inspeccionamos los 10 primeros
```

- Podemos consultar con la función `size` consultar el tamaño de los itemsets frecuentes. En este ejemplo, consultamos el tamaño, los representamos con un diagrama de barras y inspeccionamos los itemsets frecuentes de tamaño 1



```
> size(iAdult)
> barplot(table(size(iAdult)), xlab="itemset size", ylab="count")
> inspect(iAdult[size(iAdult)==1])
```

Extraer Itemsets maximales y cerrados

- Podemos son muchos itemsets frecuentes, podemos quedarnos solo con los itemsets maximales. La función `is.maximal` devuelve un vector lógico indicando que itemsets es maximal. En el ejemplo, de `iAdult` nos quedamos solo con los maximales y después mostramos los 6 primeros ordenados por su valor de soporte.

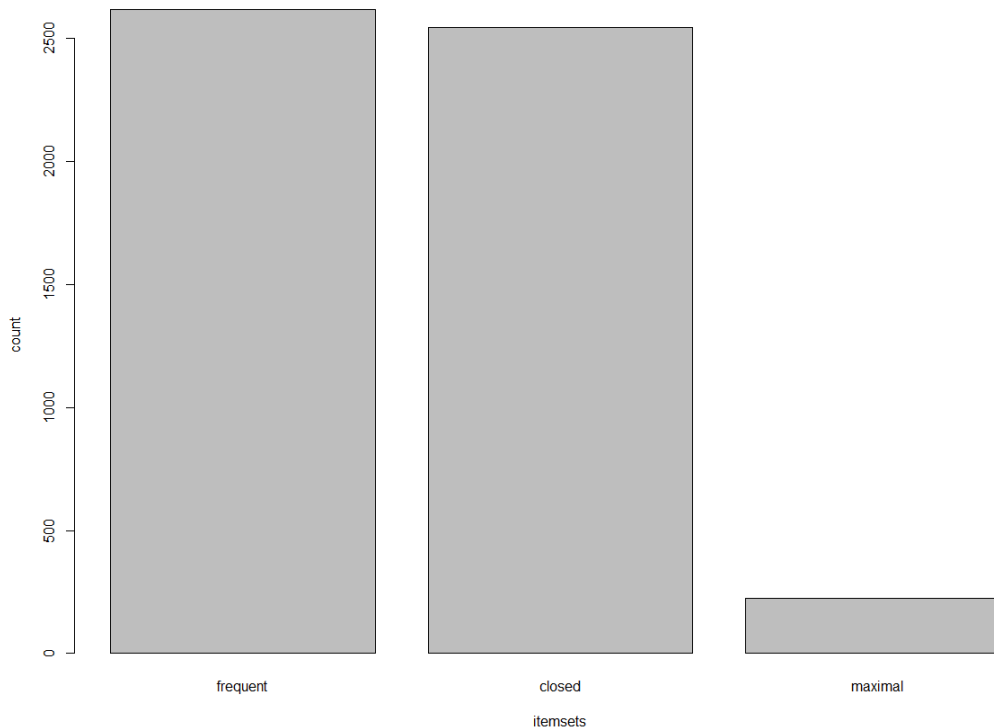
```
> imaxAdult <- iAdult[is.maximal(iAdult)]  
> inspect(head(sort(imaxAdult, by="support")))
```
- También podemos extraer los itemsets cerrados con la función `is.closed`, la cual devuelve un vector lógico indicando que itemsets es cerrado. En el ejemplo, de `iAdult` nos quedamos solo con los cerrados y después mostramos los 6 primeros ordenados por su valor de soporte.

```
> icloAdult <- iAdult[is.closed(iAdult)]  
> inspect(head(sort(icloAdult, by="support")))
```

Extraer Itemsets maximales y cerrados

- Podemos pintar un gráfico de barras para ver la cantidad de de itemsets frecuentes, cerrados y maximales que se han generado.

```
> barplot( c(frequent=length(iAdult), closed=length(icloAdult),  
maximal=length(imaxAdult)), ylab="count", xlab="itemsets")
```



Extraer reglas: Apriori

- Usamos A priori para extraer las reglas con mínimo soporte 0.1 y confianza 0.8. También indicamos que al menos la longitud de las reglas sea 2 (ant+cons):

```
> rules <- apriori(Adult, parameter = list(support = 0.1, confidence = 0.8, minlen = 2))
```

parameter specification:

confidence	minval	smax	arem	aval	originalSupport	support	minlen	maxlen	target	ext
0.8	0.1	1	none	FALSE	TRUE	0.1	2	10	rules	FALSE

algorithmic control:

filter	tree	heap	memopt	load	sort	verbose
0.1	TRUE	TRUE	FALSE	TRUE	2	TRUE

Absolute minimum support count: 4884

```
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[1 15 item(s), 48842 transaction(s)] done [0.03s].
sorting and recoding items ... [31 item(s)] done [0.01s].
creating transaction tree ... done [0.03s].
checking subsets of size 1 2 3 4 5 6 7 8 9 done [0.09s].
writing ... [6133 rule(s)] done [0.00s].
creating S4 object ... done [0.01s].
```

Información de las reglas

- Utilizamos la función *summary* para obtener información resumida del conjunto de reglas obtenido

> *summary(rules)*

set of 6133 rules

rule length distribution (lhs + rhs):sizes

2	3	4	5	6	7	8	9
121	637	1510	1903	1345	511	99	7
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.		
2.000	4.000	5.000	4.926	6.000	9.000		

summary of quality measures:

	support	confidence	lift
Min.	:0.1000	Min. :0.8004	Min. :0.9169
1st Qu.:	0.1158	1st Qu.:0.8895	1st Qu.:0.9911
Median :	0.1353	Median :0.9241	Median :1.0197
Mean :	0.1700	Mean :0.9236	Mean :1.2044
3rd Qu.:	0.1890	3rd Qu.:0.9587	3rd Qu.:1.0783
Max. :	0.8707	Max. :1.0000	Max. :2.9421

mining info:

data ntransactions	support	confidence
Adult	48842	0.1 0.8

Información de las reglas

- Podemos utilizar la función `inspect` para ver las reglas (lhs es el antecedente y rhs el consecuente de la regla) y sus valores para las medidas soporte, confianza y lift. También podemos ver solo los valores de las medidas de calidad utilizando la función `quality`.

> `inspect(head(rules))`

lhs	rhs	support	confidence	lift
1 {relationship=Unmarried}	=> {capital-loss=None}	0.1019819	0.9719024	1.0195373
2 {occupation=Sales}	=> {race=White}	0.1005282	0.8920785	1.0433144
3 {occupation=Sales}	=> {native-country=United-States}	0.1039679	0.9226017	1.0280552
4 {occupation=Sales}	=> {capital-gain=None}	0.1030670	0.9146076	0.9969706
5 {occupation=Sales}	=> {capital-loss=None}	0.1068343	0.9480378	0.9945030
6 {occupation=Adm-clerical}	=> {native-country=United-States}	0.1052373	0.9160577	1.0207632

> `quality(head(rules))`

	support	confidence	lift
1	0.1019819	0.9719024	1.0195373
2	0.1005282	0.8920785	1.0433144
3	0.1039679	0.9226017	1.0280552
4	0.1030670	0.9146076	0.9969706
5	0.1068343	0.9480378	0.9945030
6	0.1052373	0.9160577	1.0207632

Estudiar las reglas deseadas

- Ordenar las reglas por el campo que más nos interese. Por ejemplo, ordenarlas por el valor de la confianza

```
> rulesSorted = sort(rules, by = "confidence")
> inspect(head(rulesSorted))
```

- Seleccionar un subconjunto de reglas que cumplan una condición. Por ejemplo, seleccionamos las reglas que tenga lift > 1.2 y que en el consecuente de la regla tengan el itemset race=White

```
> rulesRaceWhite <- subset(rules, subset = lhs %in% "race=White" & lift > 1.2)
> inspect(head(rulesRaceWhite))
```

- Eliminar las reglas redundantes

```
> subsetMatrix <- is.subset(rulesSorted, rulesSorted)
> subsetMatrix[lower.tri(subsetMatrix, diag=TRUE)] <- FALSE
> redundant <- colSums(subsetMatrix, na.rm=TRUE) >= 1
> rulesPruned <- rulesSorted[!redundant] # remove redundant rules
> inspect(head(rulesPruned))
```

Medidas de interés adicionales

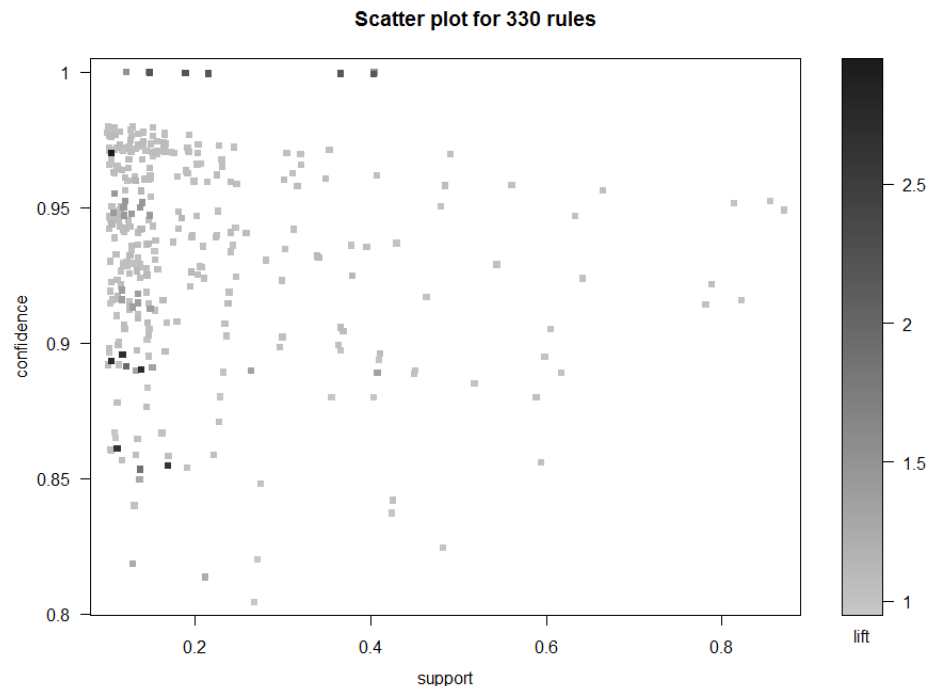
- También podemos calcular para itemsets o para reglas otras medidas de calidad utilizando la función `interestMeasure`. Ejecutar ?? `interestMeasure` para ver todas las medidas que se pueden calcular.
- Podemos calcular estas medidas para nuestras reglas podadas y añadirselas a la sección `quality` para que los valores de las medidas nuevas salgan también cuando inspeccionamos las reglas:

```
> mInteres <- interestMeasure(rulesPruned, measure=c("hyperConfidence", "leverage",
"phi", "gini"), transactions=Adult)
> quality(rulesPruned) <- cbind(quality(rulesPruned), mInteres)
> inspect(head(sort(rulesPruned, by="phi")))
```


Visualización (arulesViz)

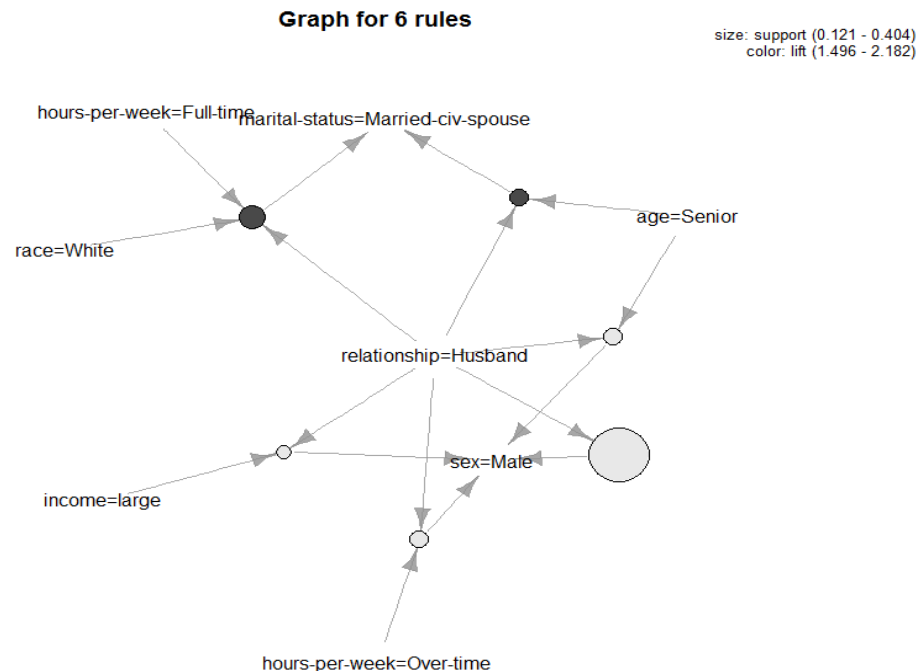
- Instalar el paquete arulesViz (versión 1.1-0) y utilizar las funciones gráficas del paquete
 - > `install.packages("arulesViz")`
 - > `library(arulesViz)`
- Utilizar la función plot para representar las reglas en función de las medidas de calidad

> `plot(rulesPruned)`



Visualización (arulesViz)

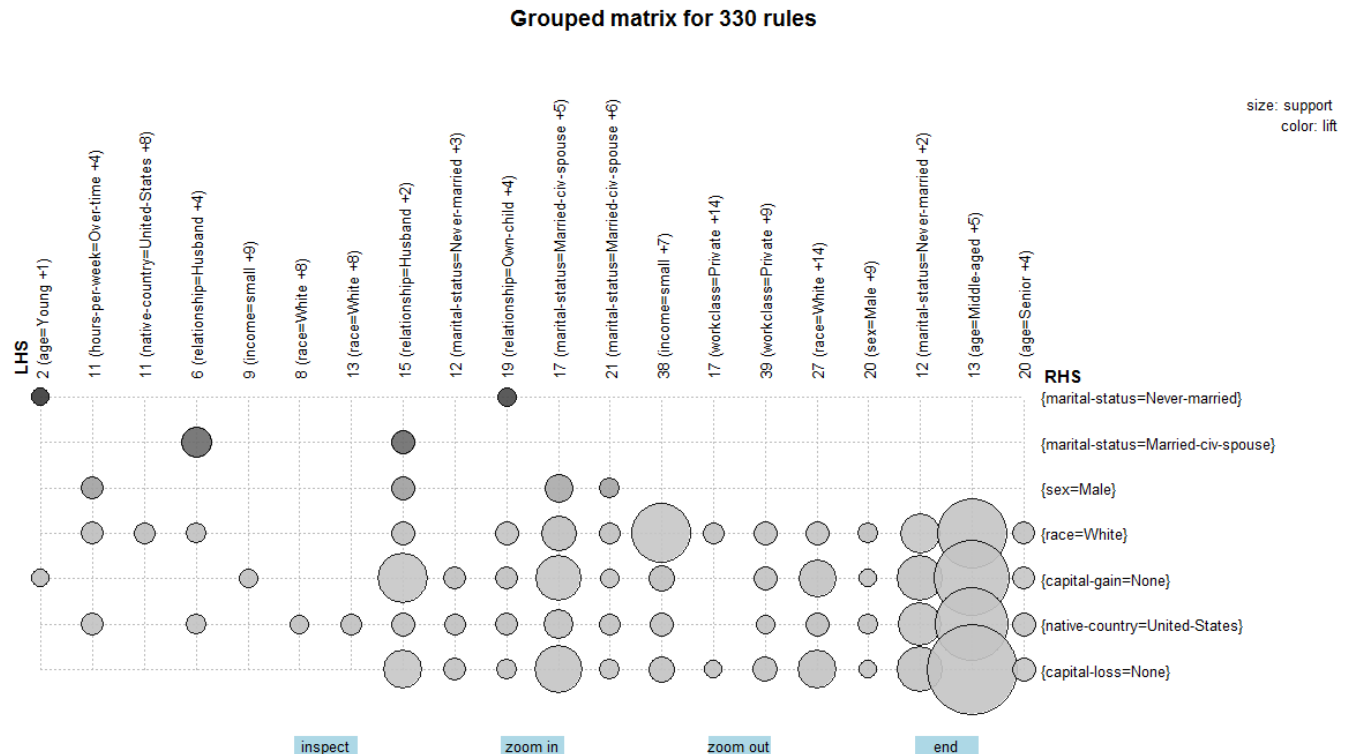
- Podemos modificar el tipo de gráfico generado cambiando el parámetro método de la función plot. Además, se puede modificar el gráfico cambiando los parámetros del tipo de gráfico seleccionado
- > `??plot` # consultar las distintas opciones para la función plot
- > `plot(rulesPruned[1:6], method="graph")`
- > `plot(rulesPruned[1:6], method="graph", engine="interactive")`



Visualización (arulesViz)

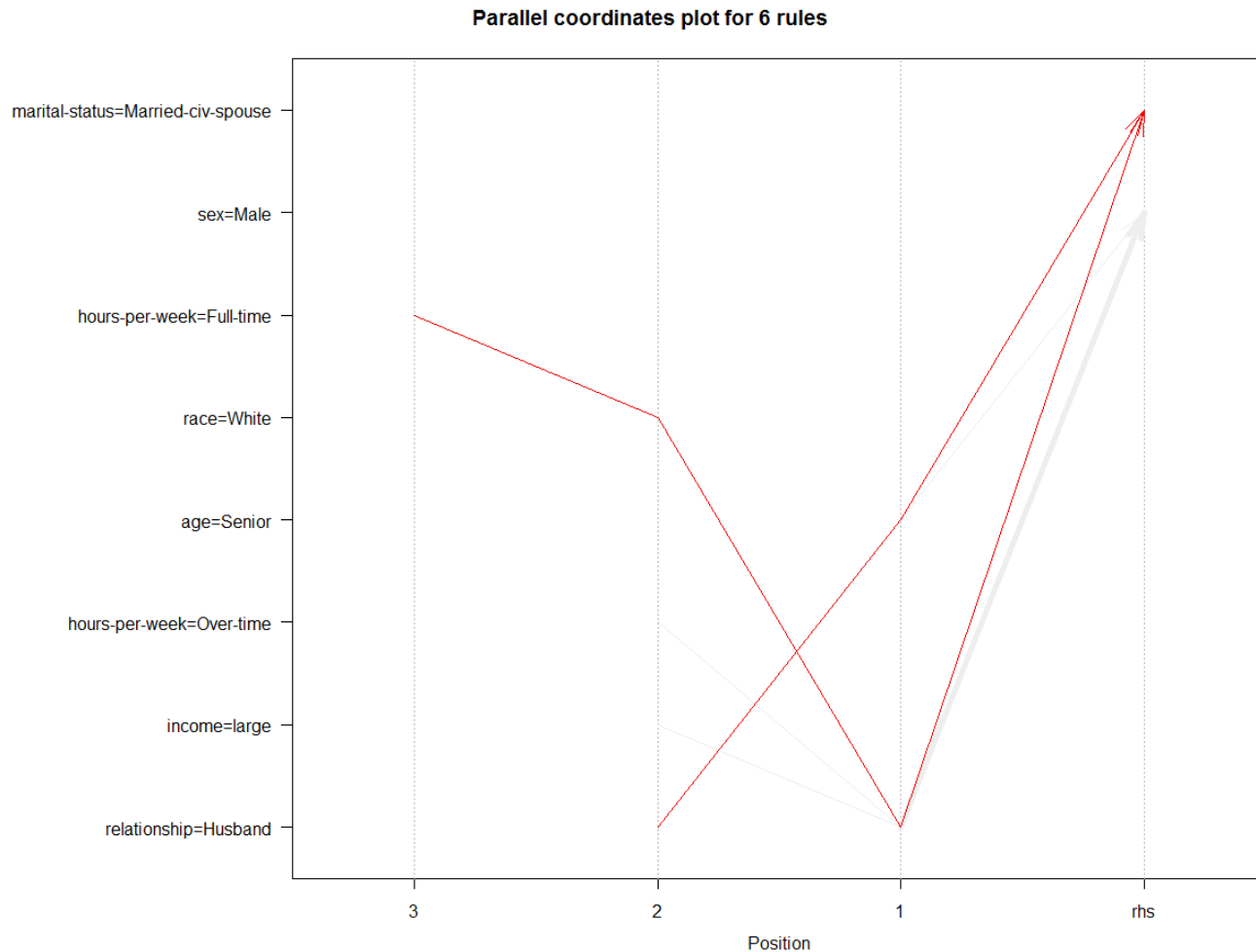
- Podemos visualizar las reglas como una matriz agrupada. Los antecedentes en las columnas son agrupados usando clustering.

```
> try: plot(rulesPruned, method="grouped")
```



Visualización (arulesViz)

```
> plot(rulesPruned[1:6], method="paracoord", reorder=TRUE)
```



Guardar y leer las reglas

- Las podemos guardar en texto plano usando la función `write`. En este ejemplo las guardamos en un fichero llamado `data.csv`, usamos como separado “,” y no le ponemos ningún nombre a las columnas
 - > `write(rulesPruned, file="reglas.csv", sep = ",")`
- También las podemos guardar en formato PMML. Para ello tendremos que tener instalado el paquete `pmml`
 - > `install.packages("pmml")`
 - > `library(pmml)`
 - > `write.PMML(rulesPruned, file="reglas.pmml")`
- Si las reglas estan guardadas en fichero en formato PMML podemos posteriormente volver a leerlas con la funcion `read.PMML`
 - > `reglasPMML = read.PMML("reglas.pmml")`

Ejercicio

- Hacer lo mismo con la BD Zoo del package mlbench. Esta es una BDs contiene la información de 7 tipos de animales diferentes de un zoo. Tiene 101 registros y 17 variables, de las cuales todas son lógicas menos una (legs) que es numérica. Para poder transformar la BD a un conjunto de transacciones, primero tendremos que dividir en intervalos esta variable. Aplicar un único corte en 0, generando 2 intervalos con las etiquetas no_legs y has_legs (tiene o no tiene piernas).
- Recordar que el paquete mlbench tiene que estar instalado previamente en RStudio (`install.packages("mlbench")`). Una vez instalado, hacemos las siguientes intrucciones para poder utilizarlo:
 - > `library(mlbench)`
 - > `data(Zoo)`
- En esta BD la mayoría de las variables son lógicas. Por lo tanto, será necesario cambiarlas todas a tipo factor antes de convertir la BD en transacciones para que el método apriori considere también los casos en los que las variables toman el valor false. Por ejemplo:
 - > `Zoo[["hair"]] <- as.factor(Zoo[["hair"]])`
- Subir a swad un fichero csv con las reglas generadas por apriori para esta BD.