PEC2 - Análisis de conglomerados

M0.517 - Análisis multivariante de datos

Juan Águila Martínez

30 de mayo de 2016

Resumen

El siguiente documento recoge los procedimientos y conclusiones de la segunda Prueba de Evaluación Continua (PEC) del curso **Análisis multivariante de datos**. Esta prueba desarrolla el análisis de conglomerados. En concreto, se desarrollan técnicas de protección de datos mediante microagregación; la microagregación consiste en la substitución de elementos del conjunto de datos original por representaciones promedio de un subconjunto de datos, para así evitar la re-identificación de individuos. El documento se estructura como sigue: en el primer apartado, se describe el objetivo del ejercicio y el entorno de software utilizado (1); dado que el conjunto de datos de esta práctica es el fichero *census.dat*, que ya se analizó en la primera PEC del curso, se evita la descripción del mismo para pasar directamente al desarrollo del ejercicio de microagregación: en primer lugar, se aplica a *census.dat* un algoritmo de k-medias para realizar la agregación con cardinalidad variable, es decir, dónde cada subconjunto puede tener distinto número de elementos (2); en segundo lugar, se desarrolla un algoritmo de agregación con cardinalidad constante, que se aplicará a los datos de *census.dat* utilizando distintos tamaños de subconjuntos (3). En ambos casos se evaluará la pérdida de información resultante de los métodos de microagregación analizando la Suma de Cuadrados Dentro de los Grupos (SCDG) en diferentes escenarios.

1. Definición de los objetivos y el entorno de trabajo

El conjunto de datos *census.dat* es un conjunto de *microdatos*, que contiene datos de carácter personal referentes a salarios, beneficios e impuestos pagados por empresas y particulares americanos durante 1995. Si bien los datos de *census.dat* no contienen referencias que puedan identificar a los titulares de dichos datos, es posible que atacantes puedan realizar una re-identificación de los individuos a partir de los datos disponibles. En este sentido, una de las técnicas más comunes para la protección de datos es la microagregación, que consiste en agrupar a los individuos del conjunto de datos original en subconjuntos disjuntos para después reemplazar las observaciones de los valores de las distintas variables por el vector de medias del subconjunto, con lo que los individuos pertenecientes a un mismo subconjunto son indistinguibles. En la microagregación, el criterio de protección se fija con la cardinalidad de los subconjuntos, de manera que, por regla general, a mayor cardinalidad mayor es la protección del individuo, pero también es mayor la pérdida de información para usos

secundarios.
En esta práctica se utilizarán dos aproximaciones complementarias a las técnicas de microagregación:

- En primer lugar, se aplicará la microagregación con cardinalidad variable -es decir, dónde cada grupo puede tener distinto número de elementos- utilizando el algoritmo de k-medias y partiendo de distinto número de subconjuntos. Para evaluar la pérdida de información, se analizará la evolución de la Suma de Cuadrados Dentro de los Grupos (SCDG) a medida que aumenta el número de grupos.

- En segundo lugar, se aplicará la microagregación con cardinalidad constante -es decir, dónde cada grupo tiene el mismo número de elementos- mediante un algoritmo desarrollado para el caso, utilizando distintos valores de cardinalidad y, por tanto, distinto número de subconjuntos. De nuevo, se analizará la SCDG a medida que aumenta la cardinalidad para evaluar la pérdida de información.

Dada la vocación práctica de este curso, la resolución del problema planteado se ha realizado con software de cálculo estadístico. A continuación se detallan las diferentes herramientas empleadas:
Para la realización de esta PEC se ha utilizado el lenguaje de programación ***R*** (v3.2.0) en el entorno de desarrollo ***RStudio*** en su versión Desktop para Mac OS X 10.6+ (64bit) (v0.99). Aprovecharemos de nuevo la amplia variedad de paquetes de funciones desarrollados para *R* con licencia *open source*, que incorporan distintas funcionalidad que descargan el proceso de análisis de la implementación de los algoritmos de cálculo. Para la realización de esta práctica se han utilizado los siguientes paquetes:

- ***pdist***: cálculo de distancias euclídeas entre los elementos de dos matrices (https://cran.r-project.org/web/packages/pdist/)

- ***ggplot2***: graficado de resultados (http://ggplot2.org/)

- ***lpSolve***: funciones para la resolución de problemas de programación lineal (https://cran.r-project.org/web/packages/lpSolve/)

- ***xTable***: generación de tablas LaTex a partir de DataFrames (https://cran.r-project.org/web/packages/xtable/)

- ***xlsx***: funciones para la lectura y escritura de datos en archivos de Microsoft Office Excel (https://cran.r-project.org/web/packages/xlsx/)

El redactado de la PEC se ha realizado con ***LaTex*** en el entorno web de ***Overleaf*** (https://www.overleaf.com/).

2. Microagregación con cardinalidad variable

2.1. El algoritmo de k-medias

El algoritmo de k-medias es un método de clásico de partición de datos que tiene como objetivo la división de un conjunto de n observaciones en G grupos en el que cada observación pertenece a uno, y sólo uno, de los grupos. El Algoritmo 1 introduce el pseudo-código del algoritmo de k-medias; la introducción de este pseudo-código nos permitirá tener una referencia cuando, en el próximo apartado, planteemos una modificación sobre el mismo para definir un algoritmo de k-medias con cardinalidad constante.

Data:
$E = \{e_1, e_2, ..., e_n\}$ (el conjunto de datos a particionar)
G (el número de particiones a realizar)
$MaxIters$ (el número de iteraciones a realizar)
Result:
$C = \{c_1, c_2, ..., c_n\}$ (el conjunto de centroides resultante)
$L = \{l(e)|e = 1, 2, ..., n\}$ (la etiqueta asociada a cada elemento del conjunto de datos)
Inicialización;
foreach c *en* C **do**
  $c_i \leftarrow e_j$ (p.e. selección aleatoria)
end
foreach e_i *en* E **do**
  $l(e_i) \leftarrow argminDistance(e_i, c_j)$ con j en $\{1, ..., n\}$
end
$changed = False$
$iter = 0$
while $changed = False$ && $iter \leq MaxIters$ **do**
  foreach c *en* C **do**
    $UpdateCluster(c_i)$
  end
  foreach e_i *en* E **do**
    $minDist \leftarrow argminDistance(e_i, c_j)$ con j en $\{1, ..., n\}$
    if $minDist \neq l(e_i)$ **then**
      $l(e_i) = minDist$ $changed = True$
    end
  end
  $iter++$
end

Algoritmo 1: algoritmo de k-medias

El criterio de optimalidad del particionado es la minimización de la Suma de Cuadrados dentro de los Grupos (SCDG), por lo que se analizará esta variable como medida de la pérdida de información.

2.2. Aplicación de k-medias a census.dat

En esta sección aplicaremos el algoritmo de k-medias *kmeans()*, que está disponible en *R* de forma nativa (sin necesidad de invocar paquetes adicionales de funciones). Cabe notar lo siguiente:

- El algoritmo *kmeans()* disponible en *R* de forma nativa utiliza *G* puntos aleatorios para inicializar los centroides si estos no se proporcionan de forma explícita
- Por defecto, el algoritmo *kmeans()* utiliza 10 iteraciones, si bien esta variable se puede pasar como argumento
- Por defecto, *kmeans()* utiliza la versión del algoritmo de k-medias de *Hartigan and Wong* (ver Algoritmo 1), si bien es posible utilizar otras versiones de k-medias definiendo el argumento *algorithm*. Las alternativas disponibles son *Lloyd* y *MacQueen*

Para facilitar la reutilización del código, se han diseñado dos funciones que aplican el algoritmo *kmeans()* obteniendo diferentes resultados:

1. La función *screePlot()* devuelve la evolución del SCDG a medida que aumenta el número de grupos
2. La función *getProperties()* devuelve las propiedades más relevantes del particionado: número de grupos, tamaño, SCDG aportado por cada grupo y SCDG total

Por último, destacar que, dado que todas las variables de *census.dat* están expresadas en las mismas unidades, se ha considerado no realizar ninguna estandarización. El Anexo 1 recoge el código *R* utilizado al completo. El Cuadro 1 y la Figura 1 muestran la evolución del SCDG total que resulta al aplicar *screePlot()* a los datos de *census.dat* con entre 1 y 15 grupos.

Número de grupos	SCDG
1	14227583263305.96
2	7047931727412.96
3	4974473310976.50
4	3988510608228.15
5	3363865640868.20
6	2895036433117.97
7	2606818974886.74
8	2386760955570.43
9	2147432824437.28
10	2094426026817.14
11	1857627875778.57
12	1733252215201.43
13	1623407519654.88
14	1593183644972.58
15	1516561707747.40

Cuadro 1: Evolución del SCDG total al variar el número de grupos entre 1 y 15

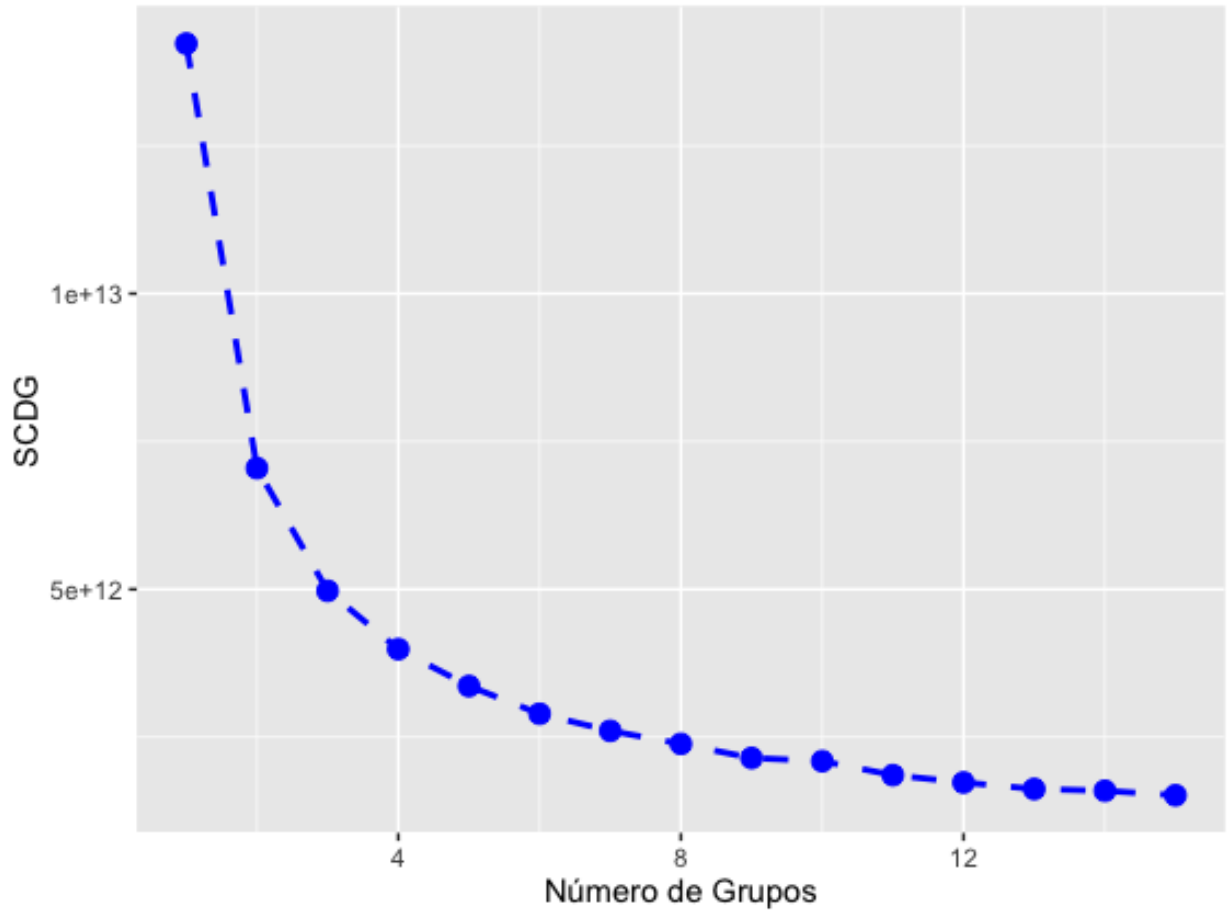

Figura 1: Evolución de la SCDG según el número de grupos en *census.dat*

El Cuadro 2-3 muestra el resultado de la aplicación de *getProperties()* a *census.dat* para un rango de 10 grupos.

	Grupos	Tamaño	SCDG(i)	SCDG (Total)
1	1.00	1080	14227583263305.96	14227583263305.96
2	2.00	263	2172549577002.01	7047931727412.96
3	2.00	817	4875382150410.95	7047931727412.96
4	3.00	436	1905017932430.68	4974473310976.50
5	3.00	164	1056456538346.28	4974473310976.50
6	3.00	480	2012998840199.55	4974473310976.50
7	4.00	295	765112423660.64	3990597359119.98
8	4.00	167	1074723953593.98	3990597359119.98
9	4.00	276	857085605888.99	3990597359119.98
10	4.00	342	1293675375976.37	3990597359119.98
11	5.00	248	855220255292.14	3366541811925.46
12	5.00	167	614684610785.61	3366541811925.46
13	5.00	263	675118624646.02	3366541811925.46
14	5.00	292	602581280818.11	3366541811925.46
15	5.00	110	618937040383.59	3366541811925.46
16	6.00	139	400252380229.08	2980776497058.36
17	6.00	139	329791879254.49	2980776497058.36
18	6.00	226	492008947338.51	2980776497058.36
19	6.00	140	827430556135.52	2980776497058.36
20	6.00	186	394035113875.64	2980776497058.36
21	6.00	250	537257620225.11	2980776497058.36
22	7.00	210	402914590706.37	2606927877206.46
23	7.00	106	592067461408.61	2606927877206.46
24	7.00	108	297080612140.85	2606927877206.46
25	7.00	243	427819507069.57	2606927877206.46
26	7.00	147	360430261779.84	2606927877206.46
27	7.00	123	254877005308.68	2606927877206.46
28	7.00	143	271738438792.53	2606927877206.46
29	8.00	124	249785123590.32	2406876790222.82
30	8.00	95	523135710011.26	2406876790222.82
31	8.00	191	322560993776.83	2406876790222.82
32	8.00	126	303739395373.81	2406876790222.82
33	8.00	163	237904313597.47	2406876790222.82
34	8.00	106	298418373213.09	2406876790222.82
35	8.00	190	344052454020.96	2406876790222.82

Cuadro 2: Evolución de las propiedades más relevantes del particionado al variar el número de grupos entre 1 y 10

	Grupos	Tamaño	SCDG(i)	SCDG (Total)
36	8.00	85	127280426639.08	2406876790222.82
37	9.00	112	189090830161.73	2218558998083.94
38	9.00	169	216539588131.33	2218558998083.94
39	9.00	117	220714618029.52	2218558998083.94
40	9.00	193	316291514807.43	2218558998083.94
41	9.00	79	137828498406.61	2218558998083.94
42	9.00	44	238841314409.98	2218558998083.94
43	9.00	121	245072437261.97	2218558998083.94
44	9.00	112	383191020713.11	2218558998083.94
45	9.00	133	270989176162.27	2218558998083.94
46	10.00	91	156989340640.66	2041362215317.24
47	10.00	145	242206575636.80	2041362215317.24
48	10.00	77	209046652048.00	2041362215317.24
49	10.00	88	142758915808.23	2041362215317.24
50	10.00	165	197508364649.83	2041362215317.24
51	10.00	163	231018408375.56	2041362215317.24
52	10.00	57	308109542239.86	2041362215317.24
53	10.00	107	188642922033.79	2041362215317.24
54	10.00	107	178292436269.50	2041362215317.24
55	10.00	80	186789057615.01	2041362215317.24

Cuadro 3: Evolución de las propiedades más relevantes del particionado al variar el número de grupos entre 1 y 15 (continuación

Para seleccionar el número de grupos de forma que se minimice la pérdida de información, se puede realizar un test F de reducción de variabilidad, comparando la SCDG con G grupos con la de G + 1, y calculando la reducción relativa de variabilidad. Utilizaremos valor de referencia empírico sugerido por Hartigan, que indica que hay que introducir un grupo adicional si el cociente es mayor que 10. Así, por ejemplo, al pasar de 2 a 3 grupos hay una reducción de variabilidad muy significativa dada por:

$$F = \frac{SCDG(G) - SCDG(G+1)}{SCDG(G+1)/(n-G-1)} = \frac{7047931727412{,}96 - 4974473310976{,}50}{4974473310976{,}50/(1080-4)} = 448{,}5$$

Pudiéndose comprobar que en ningún paso entre 1 y 15 grupos se alcanza un valor menor a 10 para el test F.

3. Microagregación con cardinalidad constante

La resolución de una partición con cardinalidad constante no se puede obtener a priori utilizando los algoritmos de k-medias habituales -ni tampoco otros algoritmos que permitan la partición, como los métodos jerárquicos. En este apartado se desarrolla una versión modificada de k-medias que, partiendo de un valor de cardinalidad k, define $G = n/k$ grupos con cardinalidad constante.

3.1. Definición del algoritmo de k-medias para cardinalidad constante

Dada una matriz de datos con un número de elementos n divisible, es posible obtener una asignación a G grupos de manera que la cardinalidad de cada grupo sea una constante $k = n/G$ como sigue:

1. En primer lugar, es necesario seleccionar $G = n/k$ centroides iniciales que permitan inicializar el proceso de partición con cardinalidad constante. Una buena alternativa es utilizar, para realizar esta asignación, el resultado del algoritmo de k-medias no modificado (con cardinalidad variable).

2. A continuación, es necesario calcular la distancia de cada punto de la matriz de datos a cada uno de los centroides obtenidos. De esta forma, se obtiene una matriz dónde cada elemento d_i, j representa la distancia de la observación i al centroide j o bien, para facilitar los cálculos posteriores, el cuadrado de dicha distancia.

3. En este punto, encontrar una asignación que reduzca la SCDG (y por tanto la varianza) pasa por encontrar una asignación de los elementos a los centroides tal que se minimice la suma de distancias d_ij sujeta a las siguientes restricciones:

   - Cada elemento sólo puede asociarse a un centroide
   - Cada centroide tiene que tener asociados un número de observaciones constante $k = n/G$

   Este problema puede ser entendido como un problema de programación lineal con variable binaria que puede ser resuelto de forma eficiente por numerosas vías[1].

4. A continuación, y para reducir la sensibilidad a la asignación inicial, se pueden calcular los centroides de los grupos obtenidos al solucionar el problema de la asignación, utilizando estos puntos como nuevos centroides iniciales. La convergencia del algoritmo esta asegurada, ya que en cada paso sólo es posible reducir la varianza.

[1] De hecho, este problema es un caso particular del típico Problema de Transporte, que puede resolverse en tiempo polinómico con numerosos algoritmos

El Algoritmo 2 recoge el pseudocódigo de este procedimiento:

Data:
$E = \{e_1, e_2, ..., e_n\}$ (el conjunto de datos a particionar)
k (el número de elementos en cada partición)
$MaxIters$ (el número de iteraciones a realizar)
Result:
$C = \{c_1, c_2, ..., c_n\}$ (el conjunto de centroides resultante)
$L = \{l(e)|e = 1, 2, ..., n\}$ (la etiqueta asociada a cada elemento del conjunto de datos)
Inicialización;
foreach c *en* C **do**
  $c_i \leftarrow e_j$ (p.e. kmeans)
end
$iter = 0$
while $iter \leq MaxIters$ **do**
  foreach e_i *en* E **do**
    $l(e_i) \leftarrow distance(e_i, c_j)$ con j en $\{1, ..., n\}$
  end
  linearProgrammingSolve(distances)
  foreach e_i *en* E **do**
    $minDist \leftarrow argminDistance(e_i, c_j)$ con j en $\{1, ..., n\}$
    foreach c *en* C **do**
      $c_i \leftarrow mean(E, groupBy = l)$ (los centroides se asocian al valor central de cada grupo)
    end
  end
  $iter++$
end

Algoritmo 2: algoritmo de k-medias modificado (cardinalidad constante)

El Anexo 1 recoge el código utilizado. Para poder resolver el problema de optimización planteado se ha utilizado el paquete de funciones *lpSolve*, que proporciona varios métodos para la resolución de problemas de programación lineal. Cabe notar que el problema se ha planteado utilizando la matriz de distancias como una matriz de costes, y utilizando una matriz auxiliar de variables de decisión que contiene, en cada posición, una variable binaria que indica si se realiza la asignación a del elemento i al centroide G o no. Las restricciones, con este planteamiento, son fáciles de expresar:

- La suma de los elementos de una fila de la matriz de variables binarias de decisión debe ser igual a 1
- La suma de los elementos de una columna de la matriz de variables binarias de decisión debe ser igual a $k = n/G$

El cuadro 4 y la Figura 2 recogen la SCDG obtenida para 3, 4, 5 y 10 grupos, y la comparativa con los resultados obtenidos con cardinalidad variable. Como era de esperar, la SCDG obtenida con el algoritmo de cardinalidad constante es superior a la obtenida con el método de k-medias habitual. Este incremento en la SCDG se debe a que, al añadir restricciones

(mismo número de elementos dentro de cada grupo) nos alejamos de la asignación óptima, que reduciría la distancia del elemento al centroide y, por tanto, la varianza (algunos puntos no pueden asignarse todo lo bien que deberían debido a las restricciones). La asignación del algoritmo de cardinalidad constante da, en consecuencia, resultados entre un 6 % y un 21 % peores que el método de cardinalidad variable para un mismo número de grupos. Cabe notar, también, que la diferencia en los resultados de ambos algoritmos se reducen a medida que se incrementa el número de grupos, debido a que el algoritmo de cardinalidad constante encuentra más posibilidades para realizar una asignación semi-óptima.

Técnica	3 grupos	4 grupos	5 grupos	10 grupos
Card. Constante (CC)	6017643992209	4597342764756	3639362546721	2166430245661
Car. Variable (CV)	4974473310976	3988510608228	3363865640868	2041362215317
SCDG CC-CV	+21 %	+15 %	+8 %	+6 %

Cuadro 4: Evolución del SCDG total al variar el número de grupos entre 1 y 15

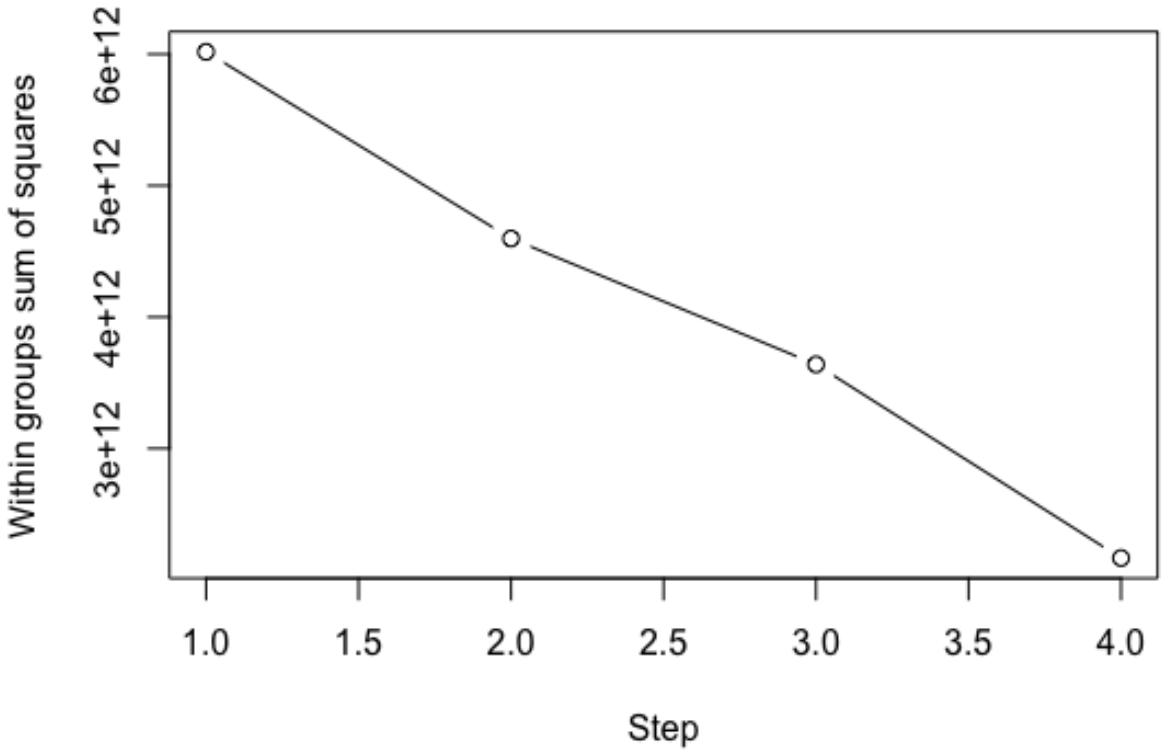

Figura 2: Evolución de la SCDG según el número de grupos en *census.dat* con cardinalidad constante

4. Anexo 1: Resumen de código utilizado

Sección 2: Particionado con cardinalidad variable

```
options(scipen=0)
install.packages("pdist")
library(pdist)
install.packages("lpSolve")
library(lpSolve)
install.packages("ggplot2")
library(ggplot2)
library(xtable)
# Carga el conjunto de datos census.dat
census <- read.csv("CASCrefmicrodata.csv", sep=",")
# Define la fn screePlot()
# Dado un conjunto de datos y un rango (2, rangeLength+1) de grupos
# Devuelve una lista con la ev. de la WSS (Within-cluster Sum of Squares)
screePlot = function(data, rangeLength=10) {
  wss = (nrow(data)-1)*sum(apply(data, 2, var))
  for (i in 2:range_length) {
    fit = kmeans(census, centers=i)
    wss[i] =sum(fit$withinss)
  }
  return(wss)
}
# Aplica la fn
rangeLength=15
wss = screePlot(census, rangeLength)
wss = data.frame(id = c(x=1:rangeLength), wss = c(wss))
# Devuelve la tabla en formato tex
<<results=tex>>
  xtable(data.frame(wss))
# Dibuja la ev. de la WSS
ggplot(data=wss, aes(x=id, y=wss, group=1)) + geom_line(colour="blue", linety

# Define la fn getProperties()
# Dado un conjunto de datos y un rango (2, rangeLength+1) de grupos
# Devuelve una tabla con las propiedades relevantes del particionado
getProperties = function(data=census, rangeLength=10) {
  wss = (nrow(data)-1)*sum(apply(data, 2, var))
  summary_data = data.frame(groups=1, size=nrow(data), wssi=wss, wss)
  for (i in 2:range_length) {
    fit = kmeans(census, centers=i)
    appendix = data.frame(groups=rep(i, i), size=fit$size, wssi=fit$withinss,
    summary_data = rbind(summary_data, appendix)
```

```
  }
  return(summary_data)
}

# Aplica la fn
summaryData = getProperties(census, range_length=10)
# Devuelve la tabla en formato tex
<<results=tex>>
  xtable(summary_data)
```

Sección 3: Particionado con cardinalidad constante

```
# 1. Dado un tamano de cluster determinado y el num. de elementos del dataset
census <- read.csv("CASCrefmicrodata.csv", sep=",")
N = nrow(census) # Num. de elementos en el dataset
n = 108 # Num. de elementos en cada grupos
G = N/n # Num. de clusters
iterations = 5 # Iteraciones
# 2. Inicializar el vector de medias de tamano n/k con un algoritmo de cluste
# Con esto obtenemos una buena estimac. de los puntos centrales de los cluste
fit = kmeans(census, centers=G)
centers = fit$centers
wss = list(fit$tot.withinss)
for(j in 1:iterations) {
  # Variante 1
  # Calcular una matriz de distancias entre cada elemento de la matriz de dat
  distances = NULL
  for(i in 1:nrow(fit$centers)) {
    distance_i = as.matrix(pdist(census[,1:13], centers[i,]))
    # Elevamos al cuadrado todos los elementos de la matriz, para simplificar
    distance_i = distance_i ^ 2
    distances[i] = data.frame(distance_i)
  }
  # Resolver el problema como un problema de OPT con restricciones
  assign.costs = matrix(unlist(distances), ncol = i, byrow = FALSE)
  row.signs <- rep ("=", N)
  row.rhs <- rep (1, N)
  col.signs <- rep ("=", i)
  col.rhs <- rep (n, i)
  clusters = lp.transport(assign.costs, "min", row.signs, row.rhs, col.signs,
  wss[j+1] = clusters$objval
  census <- data.frame(census, cluster = apply(clusters$solution,1,which.max)
  centers = aggregate(census[, 1:13], list(census$cluster), mean)[,2:14]
  # Para reducir la sensibilidad a la asign. inicial, se puede repetir este p
  # La convergencia llega porque cada cambio puede solo reducir la variancia
}
```