Optimización lineal con R

José R. Berrendero

Introducción

Veamos cómo se pueden resolver problemas de optimización lineal con **R** a través de algunos ejemplos sencillos. La mayor parte de las funciones necesarias pueden encontrarse en el paquete **lpSolve** (Berkelaar *et al*, 2014) que a su vez esta basado en el proyecto de código abierto **lp_Solve**. Otro paquete alternativo que también se puede usar es **linprog**.

El paquete *lpSolve*

Un problema en forma canónica

Como ejemplo vamos a usar el siguiente problema:

> Un pastelero dispone de 150 kg de harina, 22 kg de azúcar y 27.5 kg de mantequilla para elaborar dos tipos de pasteles (A y B). Cada caja de pasteles de tipo A requiere 3 kg de harina, 1 kg de azúcar y 1 kg de mantequilla y su venta le reporta un beneficio de 20 euros. Cada caja de pasteles de tipo B requiere 6 kg de harina, 0.5 kg de azúcar y 1 kg de mantequilla y su venta le reporta un beneficio de 30 euros.¿Cuántas cajas de cada tipo debe elaborar el pastelero de manera que se maximicen sus ganancias? (Se supone en principio que también puede elaborar cajas incompletas, es decir, que no se trata de un problema de programación entera.)

La forma canónica de un problema de optimización lineal es $\max c^\top x$ s.a. $Ax \leq b$, $x \geq 0$. La forma canónica del problema que vamos a resolver como ejemplo es:

$$\begin{aligned} \max\ & 20x_1 + 30x_2 \text{ s.a.} \\ & 3x_1 + 6x_2 \leq 150 \\ & x_1 + 0.5x_2 \leq 22 \\ & x_1 + x_2 \leq 27.5 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{aligned}$$

Para resolverlo basta llamar a la función `lp` y pasarle como argumentos de forma ordenada los parámetros que definen el problema. El orden de los parámetros es el siguiente:

1. `'min'` o `'max'` en función de si el problema es de minimización o maximización.
2. El vector de coeficientes de la función objetivo. En el ejemplo $c = (20, 30)$.
3. La matriz de coeficientes de las variables en las ecuaciones que definen las restricciones. En el ejemplo,

$$A = \begin{pmatrix} 3 & 6 \\ 1 & 0.5 \\ 1 & 1 \end{pmatrix}.$$

4. Un vector que determina la dirección de las restricciones: `<=`, `>=` o `=`. En el ejemplo las tres desigualdades son del tipo `<=`.

5. El vector con los términos independientes de las restricciones. En el ejemplo $b = (150, 22, 22.5)$.

Por defecto se supone que las variables de decisión no pueden ser negativas.

El siguiente código define todos los parámetros del problema y se los pasa como argumentos a la función `lp` en el orden que hemos indicado:

```
library(lpSolve)

# Parametros del problema
coef <- c(20, 30)
A <- matrix(c(3, 1, 1, 6, 0.5, 1), ncol=2)
b <- c(150, 22, 27.5)
dir <- rep('<=', 3)

# Solucion
solucion <- lp('max', coef, A, dir, b)
```

El objeto `solucion` que hemos generado al ejecutar la última línea es una lista con diversos elementos entre los cuáles los más importantes son `objval`, el valor objetivo óptimo, y `solution`, las coordenadas de la solución factible óptima del problema:

```
solucion$objval
```

```
## [1] 775
```

```
solucion$solution
```

```
## [1]  5.0 22.5
```

Esto significa que la producción óptima del pastelero es de 5 cajas de A y 22.5 cajas de B, con lo que tendrá el máximo beneficio posible de 775 euros.

Programación entera

Supongamos ahora que el pastelero solo puede fabricar un numero entero de cajas de cada tipo, es decir, que las variables x_1 y x_2 solo pueden tomar valores enteros. La forma de indicar esta restricción adicional es añadiendo en la función `lp` el argumento `all.int=TRUE` (también existe la posibilidad de restringir únicamente un subconjunto de las variables):

```
solucion <- lp('max', coef, A, dir, b, all.int=TRUE)
solucion$objval
```

```
## [1] 770
```

```
solucion$solution
```

```
## [1]  4 23
```

En este caso, la producción óptima del pastelero es de 4 cajas de A y 23 cajas de B, y el beneficio de 770 euros.

Problema de transporte

El paquete **lpSolve** incluye también la función `lp.transport` para resolver el problema de transporte. Como ejemplo, vamos a resolver el problema de transporte definido por los siguientes datos:

3	4	6	8	9	30
2	2	4	5	5	80
2	2	2	3	3	10
3	3	2	4	2	60
10	50	20	80	20	

Las filas corresponden a los lugares de origen y las columnas a los lugares de destino. Los valores de la matriz son los costes de transporte desde cada origen a cada destino. El último elemento de cada fila es la oferta de producto en cada origen. El último elemento de cada columna es la demanda total en cada destino.

A continuación definimos todos los parámetros del problema y los pasamos como argumentos de `lp.transport` en el orden adecuado:

```
cost <- matrix(c(3, 2, 2, 3, 4, 2, 2, 3, 6, 4, 2, 2, 8, 5, 3, 4, 9, 5, 3, 2), ncol=5)
direction <- 'min'
row.signs <- rep('=', 4)
row.rhs <- c(30, 80, 10, 60)
col.signs <- rep('=', 5)
col.rhs <- c(10, 50, 20, 80, 20)
lp.transport(cost, direction, row.signs, row.rhs, col.signs, col.rhs)
```

```
## Success: the objective function is 610
```

```
lp.transport(cost, direction, row.signs, row.rhs, col.signs, col.rhs)$solution
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   10   20    0    0    0
## [2,]    0   30    0   50    0
## [3,]    0    0    0   10    0
## [4,]    0    0   20   20   20
```

La solución óptima del problema es transportar 10 desde el origen 1 hasta el destino 1, 20 desde el origen 1 hasta el destino 2, 30 desde el origen 2 hasta el destino 2, etc. Esta solución óptima conlleva un coste de 610.

Problema de asignación

Para resolver el problema de asignación el paquete **lpSolve** incluye la función `lp.assign`. Como ejemplo, vamos a resolver el problema de asignación definido por la matriz:

$$\begin{pmatrix} 15 & 10 & 9 \\ 9 & 15 & 10 \\ 10 & 12 & 8 \end{pmatrix}$$

Las filas corresponden a los trabajadores y las columnas a las tareas. Cada elemento de la matriz representa el coste de asignar la tarea i al trabajador j.

A continuación definimos la matriz de costes, que es el unico argumento de `lp.assign`:

```r
cost <- matrix(c(15, 9, 10, 10, 15, 12, 9, 10, 8), nrow=3)
lp.assign(cost)
```

```
## Success: the objective function is 27
```

```r
lp.assign(cost)$solution
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    1    0    0
## [3,]    0    0    1
```

La solución óptima es asignar la tarea 1 al trabajador 2, la tarea 2 al trabajador 1 y la tarea 3 al trabajador 3. El coste de esta solución es 27.

El paquete *linprog*

Una alternativa a la función `lp` de **lpSolve** es la función `solveLP` del paquete **linprog**. Su funcionamiento es similar, aunque el rango de problemas que puede resolver es menor. Por ejemplo, no trata adecuadamente problemas en forma estándar con restricciones de igualdad.

El orden por defecto en el que hay que pasar los argumentos es diferente al de `lp`:

1. Coeficientes de la función objetivo c.
2. Términos independientes de las restricciones b.
3. Matriz de coeficientes de las variables en las restricciones A.
4. Determinar si queremos maximizar o minimizar (`maximum=TRUE` o `maximum=FALSE`).
5. Vector que determina la dirección de las restricciones: `<=`, `>=` o `=`.

```r
library(linprog)
# Parametros del problema
coef <- c(20, 30)
A <- matrix(c(3, 1, 1, 6, 0.5, 1), ncol=2)
b <- c(150, 22, 27.5)
dir <- rep('<=', 3)

# Solucion
solucion <- solveLP(coef, b, A, maximum=TRUE, dir)
summary(solucion)
```

```
##
##
## Results of Linear Programming / Linear Optimization
##
## Objective function (Maximum): 775
##
## Solution
##    opt
## 1  5.0
## 2 22.5
```

La función `solveLP` tiene la ventaja sobre `lp` de dar más información sobre los pasos intermedios del algoritmo simplex utilizado para resolver el problema. Para el máximo nivel de detalle se usa la opción `verbose=4`.

```
solveLP(coef, b, A, maximum=TRUE, dir, verbose=4)
```

```
## [1] "initial Tableau"
##       1     2 S 1 S 2 S 3    P0
## 1     3   6.0   1   0   0 150.0
## 2     1   0.5   0   1   0  22.0
## 3     1   1.0   0   0   1  27.5
## Z-C -20 -30.0   0   0   0   0.0
##
## Pivot Column: 2 ( 2 )
## Pivot Row: 1 ( 1 )
##
##         1 2      S 1 S 2 S 3    P0
## 2    0.50 1  0.16667   0   0  25.0
## 2    0.75 0 -0.08333   1   0   9.5
## 3    0.50 0 -0.16667   0   1   2.5
## Z-C -5.00 0  5.00000   0   0 750.0
##
## Pivot Column: 1 ( 1 )
## Pivot Row: 3 ( 3 )
##
##     1 2     S 1 S 2  S 3     P0
## 2   0 1  0.3333   0 -1.0  22.50
## 2   0 0  0.1667   1 -1.5   5.75
## 1   1 0 -0.3333   0  2.0   5.00
## Z-C 0 0  3.3333   0 10.0 775.00

##
##
## Results of Linear Programming / Linear Optimization
##
## Objective function (Maximum): 775
##
## Iterations in phase 1: 0
## Iterations in phase 2: 2
## Solution
##    opt
## 1  5.0
## 2 22.5
##
## Basic Variables
##       opt
## 1    5.00
## 2   22.50
## S 2  5.75
##
## Constraints
##   actual dir  bvec free     dual dual.reg
## 1 150.00  <= 150.0 0.00  3.33333    34.50
## 2  16.25  <=  22.0 5.75  0.00000     5.75
```

```
## 3  27.50  <=  27.5 0.00 10.00000     2.50
##
## All Variables (including slack variables)
##       opt cvec min.c    max.c      marg marg.reg
## 1    5.00   20    15 30.00000        NA       NA
## 2   22.50   30    20 40.00000        NA       NA
## S 1  0.00    0  -Inf  3.33333  -3.33333     34.5
## S 2  5.75    0   -20  6.66667   0.00000       NA
## S 3  0.00    0  -Inf 10.00000 -10.00000      2.5
```

Algunas observaciones sobre estos resultados:

- La última fila está cambiada de signo respecto a como aparece en la mayoría de los libros de programación lineal, es decir, aparece $c_j - z_j$ en lugar de $z_j - c_j$.
- Aunque en el ejemplo anterior no se observa, en general no entra en la base la variable con el valor más negativo de $c_j - z_j$. Se utiliza otro criterio (véase el primer ejercicio).
- La columna de la derecha contiene los valores $\bar{b} = B^{-1}b$. El último elemento de esta columna da el valor objetivo para la base correspondiente a cada iteración, es decir, $c_B^\top \bar{b}$.

Ejercicios

1. Resuelve el siguiente problema de programación lineal para los valores $\theta \in \{2.9, 3, 3.1\}$:

$$\begin{array}{ll} \text{Maximizar} & x_1 + \theta x_2 \\ \text{sujeto a} & \\ & 2x_1 + 3x_2 \leq 6 \\ & -x_1 + x_2 \leq 1 \\ & x_i \geq 0,\ i = 1, 2 \end{array}$$

Usa el paquete **linprog** de forma que obtengas la máxima información posible sobre las distintas iteraciones del método simplex. Observa qué variable entra en la base en la primera iteración y trata de deducir el criterio de entrada.

2. Resuelve el siguiente problema de optimización lineal:

$$\begin{array}{ll} \text{Minimizar} & x_2 - 3x_3 + 2x_5 \\ \text{sujeto a} & \\ & x_1 + 3x_2 - x_3 + 2x_5 = 7 \\ & -2x_2 + 4x_3 + x_4 = 12 \\ & -4x_2 + 3x_3 + 8x_5 + x_6 = 10 \\ & x_i \geq 0,\ i = 1, \ldots, 6 \end{array}$$

Usa el paquete **linprog** de forma que obtengas la máxima información posible sobre las distintas iteraciones del método simplex.

3. Resuelve usando el paquete **lpSolve** el siguiente problema en el que las variables x_1 y x_2 solo pueden tomar valores enteros. Compara la solución con la que se obtiene si no se fuerza a las variables a tomar valores enteros.

$$\begin{array}{ll} \text{Maximizar} & x_1 + x_2 \\ \text{sujeto a} & \\ & -2x_1 + 2x_2 \leq 1 \\ & 16x_1 - 14x_2 \leq 7 \\ & x_i \geq 0,\ i = 1, 2 \end{array}$$

4. Resuelve el siguiente problema de transporte:

10	5	6	7	25
8	2	7	6	25
9	3	4	8	50
15	20	30	35	

5. Escribe un programa que utilice técnicas de optimización lineal para, dada una muestra de datos $y_1, \ldots, y_n$, encontrar el valor de θ que minimiza $\sum_{i=1}^{n} |y_i - \theta|$.

Referencias

Berkelaar, M. *et al* (2014) *lpSolve: Interface to Lpsolve v. 5.5 to solve linear-integer programs.* R package version 5.6.10.