Mínimos cuadrados alternados con regularización para un sistema de recomendación de Netflix.

Equipo 13

183340 Dante Ruiz
144089 Laura López S.Jacome

4/29/2019

Contents

Problema:

La misión de Netflix es conectar a las personas con las películas que aman. Para ayudar a los clientes a encontrar esas películas, Netflix ha desarrollado un sistema de recomendación de películas de clase mundial: CinematchSM. Su trabajo es predecir si alguien disfrutará de una película en función de cuánto les haya gustado o no a otras películas. Usamos esas predicciones para hacer recomendaciones personalizadas de películas basadas en los gustos únicos de cada cliente .

Objetivo:

Utilizar los ratings de diferentes usuarios para películas, para predecir el rating el resto de las películas que no han visto.

Programar en de manera local en R el algorítmo de mínimos cuadarados alternados que se ha utilizado para ajustar sistemas de recomendción, utilizando los datos de la competencia de Netflix. Asimismo, comparar su desempeño con otras implementaciones disponibles.

Datos:

Los datos provienen de la competencia de Netflix CinematchSM que consisten en un conjunto de entrenamiento y un conjunto de prueba.

El **conjunto de entrenamiento** contiene más de 100 millones de ratings que corresponden a 480,000 usuarios y 18,000 películas.

Cada registro en el conjunto de entrenamiento contiene:

```
dat_netflix <- read_csv( "datos/dat_muestra_nflix.csv", progress = FALSE) %>%
  select(-usuario_id_orig) %>%
  mutate(usuario_id = as.integer(as.factor(usuario_id)))
```

```
## Parsed with column specification:
## cols(
##   peli_id = col_double(),
##   usuario_id_orig = col_double(),
##   calif = col_double(),
##   fecha = col_date(format = ""),
##   usuario_id = col_double()
## )
```

```
head(dat_netflix)
```

```
## # A tibble: 6 x 4
##   peli_id calif fecha      usuario_id
##     <dbl> <dbl> <date>          <int>
## 1       1     3 2004-04-14          1
## 2       1     3 2004-12-28          2
## 3       1     4 2004-04-06          3
## 4       1     4 2004-03-07          4
## 5       1     4 2004-03-29          5
## 6       1     2 2004-07-11          6
```

- user
- movie
- date
- rating: calificaciones categóricas ordinales del 1 al 5.

El conjunto de prueba contiene 2.8 millones de registros y no contiene ratings.

Adicionalmente, se proporcionó un catálogo de películas:

```
pelis_nombres <- read_csv('datos/movies_title_fix.csv', col_names = FALSE, na = c("", "NA", "NULL"))
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_character()
## )
```

```r
names(pelis_nombres) <- c('peli_id','año','nombre')
head(pelis_nombres)
```

```
## # A tibble: 6 x 3
##   peli_id   año nombre
##     <dbl> <dbl> <chr>
## 1       1  2003 Dinosaur Planet
## 2       2  2004 Isle of Man TT 2004 Review
## 3       3  1997 Character
## 4       4  1994 Paula Abdul's Get Up & Dance
## 5       5  2004 The Rise and Fall of ECW
## 6       6  1997 Sick
```

En el trabajo utilizaremos una muestra de los datos de Netflix que contiene 20 millones de ratings.

Algunas problemáticas de los datos son:

1. Tamaño de los datos. Mayor tiempo de entrenamiento y requerimiento en la memoria
2. Matriz rala. Solo 1% de la matriz de usuarios películas ha sido observada, con la mayoría de las entradas sin registar.
3. Ruido. No todos los usuarios califican con precisión y todos tienen una percepción de cómo calificar las películas.
4. Distribución temporal de los ratings. El conjunto de entrenamiento abarca los ratings del periodo 1995-2005. El conjunto de prueba abarca ratings en el periodo 2006.
5. Usuarios con pocas calificaciones. Es díficil predecir con precisión los ratings de usuarios que con pocas calificaciones apenas y aparecen en el conjunto de entrenamiento y de prueba.

Métodos:

Métodos colaborativos

- Memory based techniques
- User based collaborative filtering
- Item-based collaborative filtering

Model-based techniques

- Principal component analysis (PCA)
- Probabilistic Matrix Factorization (PMF)
- SVD
- Mínimos cuadrados alternados

Content-based filtering

- Term-frequency-inverse document frecuency (TF - IDF)
- Probabilistic methods

Hypbrid filtering

Mínimos cuadrados alternados

Supongamos entonces que queremos encontrar matrices U y V, donde U es una matrix de $n \times k$ ($n =$ número de usuarios), y V es una matriz de $p \times k$, donde p es el número de películas que nos de una aproximación de la matrix X de calificaciones.

$$R \approx UM^t$$

Ahora supongamos que conocemos V_1. Si este es el caso, entonces queremos resolver para U_1:

$$\min_{U_1} ||R - U_1 M_1^t||_{obs}^2$$

Como V_1^t están fijas, este es un problema de mínimos cuadrados usual, y puede resolverse analíticamente (o usar descenso en gradiente, que es simple de calcular de forma analítica) para encontrar U_1. Una vez que encontramos U_1, la fijamos, e intentamos ahora resolver para V:

$$\min_{V_2} ||R - U_1 M_2^t||_{obs}^2$$

Y una vez que encontramos V_2 resolvemos

$$\min_{U_2} ||R - U_2 M_2^t||_{obs}^2$$

Continuamos este proceso hasta encontrar un mínimo local o hasta cierto número de iteraciones. Para inicializar V_1, se recomienda tomar como primer renglón el promedio de las calificaciones de las películas, y el resto números aleatorios chicos (por ejemplo $U(0, 1)$). También pueden inicializarse con números aleatorios chicos las dos matrices.

Intuición:

La intuición del algorítmo de mínimos cuadrados alternados es que dada una matriz, R, done las columnas son películas y los renglones son usuarios y cada entrada de la matriz son los ratings que da un usuario i para una película j.

La naturaleza del problema implica que la matriz R tendrá muchas entradas sin calificaciones ya que no todos los usuarios han visto y por ende calificado todo el universo de películas. En ese sentido la matriz R es una matriz rala.

El objetivo del problema es predecir ratings razonables para estas entradas vacias por usuario y película. Con el método de ALS esto se hace aproximando R como una descomposición de matrices U * M, donde U es la matriz de usuarios y M es una matriz de películas.

La matriz U tiene dimensiones $n_u \times n_f$ donde n_u, número de usuarios y n_f el número de factores.

La matriz M tiene dimensiones $n_f \times n_m$ donde, n_f el número de factores y n_u, número de de películas.

Intuitivamente se entiende que la matriz M describe cuánto puntaje cada película según los factores latentes. Estos factores latentes se pueden interpretar en el caso de películas cómo géneros de película tipo comedia, terror, acción, etc, que encuentre la descomposición matricial.

Por otro lado, la matríz U describe que tanto le gusta al usuario ese género de películas (factores latentes).

En ese sentido, la descomposición matricial de R en U x M se puede resolver cómo un problema de optimización donde la función objetivo a minimizar es:

$$\min_{U,V} f(U,M) = \sum_{(i,j)\, obs} w_{i,j}(r_{ij} - u_i^t m_j)^2 + \lambda \left(\sum_i n_{u_i} ||u_i||^2 + \sum_j n_{m_j} ||m_j||^2 \right)$$

El primer término a la derecha minimiza $R - U \times M$ que significa error entre la matriz observada y su aproximación. Para solo considerar los ratings originales de los usuarios se utiliza la matriz de pesos W que contiene los elementos $w_{i,j}$ que toma el valor de 1 si el rating original era conocido y 0 cuando no se conocía la película o el rating. De este modo W garantiza que solo se tengna en cuenta las calificaciones conocidas durante la minimización de costo.

El segundo término es la regularización tipo ridge para lambda grandes este término garantiza que la matriz U y M no sean demasiado grandes evitando el sobreajuste de los datos.

n_{u_i} es el número de ratings disponibles para el usuario u_i

n_{m_j} es el número de ratings disponibles para la película m_j

Métrica de desempeño

La métrica que se utilizó en el concurso y que se utilizará también en este proyecto es la RMSE (root mean squared error). El objetivo del algorítmo es minimizar esta función de pérdida.

```
recm <- function(calif, pred){
  sqrt(mean((calif - pred)^2))
}
```

Algorítmo del método a implementar

El enfoque de este proyecto en particular es utilizar alguno de los enfoque de técnicas de modelación. En particular nos gustaría utilizar el Mínimos cuadrados alternados para resolver el problema.

Este método es un filtro colaborativo para generar recomendaciones. Estos métodos usan gustos o intereses de usuarios/artículos similares — en el sentido de que les han gustado los mismos artículos/les gustaron a las mismas personas.

Ejemplo: Me gustó StarWars y Harry Potter, varios otros usuarios a los que también les gustaron estas dos películas también les gustó “Señor de los anillos”, así que recomendamos “Señor de los Anillos”.

A fin de lograr una implementación exitosa del modelo, primero utilizaremos una simulación de los datos de Netflix para tener una matriz de usuarios, películas y calificaciones pequeña y manejable. Posteriormente probaremos con los datos de la competencia.

Simulamos datos de ratings para peliculas y usuarios.

```
n_usuarios <- 100

starwars1 <- sample.int(6, n_usuarios, replace = TRUE) - 1
starwars2 <- sample.int(6, n_usuarios, replace = TRUE) - 1
starwars3 <- sample.int(6, n_usuarios, replace = TRUE) - 1
harrypotter1 <- sample.int(6, n_usuarios, replace = TRUE) - 1
harrypotter2 <- sample.int(6, n_usuarios, replace = TRUE) - 1
harrypotter3 <- sample.int(6, n_usuarios, replace = TRUE) - 1
legallyblond <- sample.int(6, n_usuarios, replace = TRUE, prob = c(.5,.1,.1,.05,.03,.02)) - 1
little_mermaid <- sample.int(6, n_usuarios, replace = TRUE, prob = c(.5,.1,.1,.1,.1,.1)) - 1
mamamia <- sample.int(6, n_usuarios, replace = TRUE, prob = c(.5,.1,.1,.1,.1)) - 1
```

```
usuario_id <- seq(1, n_usuarios)


sample_matrix <- cbind(usuario_id, starwars1, starwars2, starwars3, harrypotter1, harrypotter2, harrypo

sample_df <- as_data_frame(sample_matrix)
```

```
## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

Creamos un catalogo con los ids de las películas

```
catalogo_pelis <- as_data_frame(cbind(names(sample_df),seq(1,10)))
names(catalogo_pelis) <- c("nombre_peli", "peli_id")
catalogo_pelis
```

```
## # A tibble: 10 x 2
##    nombre_peli    peli_id
##    <chr>          <chr>
##  1 usuario_id     1
##  2 starwars1      2
##  3 starwars2      3
##  4 starwars3      4
##  5 harrypotter1   5
##  6 harrypotter2   6
##  7 harrypotter3   7
##  8 legallyblond   8
##  9 little_mermaid 9
## 10 mamamia        10
```

Creamos una matriz rala en la forma de un dataframe, usuario_id y peli_id son las coordenadas de los ratings en la matriz rala.

```
muestra_netflix <- sample_df %>% gather(nombre_peli,rating, 2:10) %>% filter(rating >0) %>% left_join(ca
```

```
## Joining, by = "nombre_peli"
```

Convertimos el dataframe al formato de matriz rala indicando los ejes. x = usuarios, y = películas con las siguientes dimensiones.

```
# Convertimos los datos a una matrix de formato: usarios X peliculas = R
R <- sparseMatrix(
  as.integer(muestra_netflix$usuario_id), # renglón
  as.integer(muestra_netflix$peli_id),    # columna
  x = muestra_netflix$rating)              # rating

# Obtenemos las dimensiones de la matriz R
n_u <- dim(R)[1]
n_m <- dim(R)[2]

paste("Tenemos una matriz rala de dimensiones ", n_u, " x ", n_m, sep="")
```

```
## [1] "Tenemos una matriz rala de dimensiones 100 x 10"
```

Las primeras 10 entradas de la matriz se ven de la siguiente forma:

```
head(R,10)
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"
##
##  [1,] . . 4 3 4 4 5 . . .
##  [2,] . 1 2 . 4 5 4 . 4 3
##  [3,] . 5 2 3 4 4 2 . . .
##  [4,] . 1 4 2 3 4 5 . . 4
##  [5,] . . 4 . 3 2 3 . 1 .
##  [6,] . 5 3 1 3 4 3 5 . 3
##  [7,] . 1 4 . 1 . 4 . 5 1
##  [8,] . 1 . 5 3 . 5 . 4 .
##  [9,] . 4 4 . . 1 4 3 . .
## [10,] . . 5 1 3 1 1 1 4 .
```

Obtenemos matriz de pesos W que es una variable indicadora de si hay o no calificación para la película, 1 y 0 respectivamente. Esta índicadora será de utilidad para calcular el error de aproximación. En el paper Yunhong .Z et al 2008 está matriz se llama I.

```
# Obtenemos matriz de pesos W

# La matriz W tiene 0 (NA) cuando el usuario no ha calificado la película, y tiene 1 para las películas
W <- R
W@x[!is.na(W@x)] <- 1

head(W)
```

```
## 6 x 10 sparse Matrix of class "dgCMatrix"
##
## [1,] . . 1 1 1 1 1 . . .
## [2,] . 1 1 . 1 1 1 . 1 1
## [3,] . 1 1 1 1 1 1 . . .
## [4,] . 1 1 1 1 1 1 . . 1
## [5,] . . 1 . 1 1 1 . 1 .
## [6,] . 1 1 1 1 1 1 1 . 1
```

Especificamos la función objetivo del problema de optimización a resolver que especificamos en la sección de intuición.

```
# Función de costo
##  n_u_i: número de ratings para cada usuario
##  n_m_j: número de ratings para cada pelicula

fn_costo <- function (R, U, M, W, lambda, n_u_i, n_m_j) {
  sum(W * ((R - (U %*% M)) ^ 2)) + lambda * (sum(n_u_i %*% (U ^ 2)) + sum((M ^2) %*% n_m_j))
}
```

Damos valores a los parámetros de regularización λ, números de factores latentes, y número de iteraciones. Asímismo fijamos una semilla para la reproducibilidad del modelo.

```r
# Semilla
set.seed(28882)
# Regularización
lambda = 0.1
# Número de factores latentes
n_factors = 5
# Número de iteraciones
n_iterations = 100
```

Inicializamos la matriz de películas $M = factoresXpeliculas$ se incializará con números aleatorios pequeños provenientes de una distribución uniforme. δ es un parámetro que tiene solo el propósito de hacer los números chicos.

En el artículo de Yunhong .Z et al 2008, se sugiere inicializar el primer factor latente (primer renglón) de la matriz de M utilizando los promedios de las calificaciones para cada película. En caso de que exista alguna película sin calificación se inicializa con el promedio de todas las películas.

Inicializamos U con 1s y 0s aunque en realidad no importan los valores ya que la matriz U se va a sobreescribir en la primera fase del ciclo

```r
# Factor para hacer chicos los números.
delta <- 0.0001
# Inicializamos la matriz de películas y factores latentes
M <- delta * matrix(runif(n_m * n_factors),nrow = n_factors, ncol = n_m)

# Como se sugiere en el artículo, se inicializará la primera fila de la matriz M con los promedios de c
M[1, ] <- colSums(R, na.rm = TRUE) / colSums(W)
calif_promedio <- mean(M[1, ], na.rm = TRUE)
M[1, ][is.na(M[1, ])] <- calif_promedio

# Inicializamos la matriz de Us con 1.
U <- matrix(0, nrow = n_u, ncol = n_factors)
U[, 1] <- 1

# Número de ratings para cada usuario
n_u_i <- rowSums(W)

# Número de usuarios para cada pelicula
n_m_j <- colSums(W)
```

A continuación calculamos el valor de la función objetivo en la iteración cero. Es decir solo sustituimos los valores iniciales en la función de costo.

```r
# Costo de la primera iteración
cost <- fn_costo(R, U, M, W, lambda = lambda, n_u_i, n_m_j)

print(paste0("Iteración 0: Función de costo = ", cost))
```

```
## [1] "Iteración 0: Función de costo = 1914.97968804427"
```

A continuación proponemos el algorítmo para actualizar la función objetivo actualizando los valores de U y M. En cada iteración primero se fija M_0 para obtener U_1, después se usa esa U_1 para calcular M_1 y se sustituyen estos valores en la función objetivo para calcular al pérdida. De esta manera esta iteración actualiza los valores de la matriz de usuarios y películas.

```r
# Este for aproxima U%*%M a R, considerando la matriz de pesos W para calcular el error
for (kk in 1:n_iterations) {
  # El primer paso es fijar M y minimizar la fn de costo. Se itera sobre los usuarios

  for (ii in 1:n_u) {
    # Primero hay que desechar las columnas M y filas R que son irrelevantes para
    #el usuario ii, para poder incrementar la velocidaddel algorítmo

    M_selected <- M[, W[ii,] == 1, drop = FALSE]
    R_selected <- R[ii,][W[ii,] == 1, drop = FALSE]
    # Actualiza U para cada usuario ii
    U[ii,] <-
      t(solve(
        M_selected %*% t(M_selected) + lambda * n_u_i[ii] * diag(n_factors),
        (M_selected %*% R_selected)
      ))
  }
  cost <- fn_costo(R, U, M, W, lambda = lambda, n_u_i, n_m_j)
  print(paste0(kk, "iteración, paso 2: Función de costo = ", cost))

  # Minimizar U %*% M para una U fija iterando sobre n usuarios (items)
  for (jj in 1:n_m) {
    if (sum(W[, jj] == 1) > 0) {
      U_selected <- U[W[, jj] == 1, , drop = FALSE]
      R_col_selected <- R[, jj][W[, jj] == 1, drop = FALSE]
      # Actualiza M para cada película jj
      M[, jj] <-
        solve(
          t(U_selected) %*% U_selected + lambda * n_m_j[jj] * diag(n_factors),
          t(U_selected) %*% R_col_selected
        )
    }
  }
  cost <- fn_costo(R, U, M, W, lambda = lambda, n_u_i, n_m_j)
  print(paste0(kk, "iteración, paso 2: Función de costo = ", cost))
}
```

```r
## vector de kk
vector_kk <- vector(mode = "numeric", length = n_iterations)
vector_kkk <- vector(mode = "numeric", length = n_iterations)
vector_kkkk <- vector(mode = "numeric", length = n_iterations)


# Este for aproxima U%*%M a R, considerando la matriz de pesos W para calcular el error
for (kk in 1:n_iterations) {
  # El primer paso es fijar M y minimizar la fn de costo. Se itera sobre los usuarios

  for (ii in 1:n_u) {
    # Primero hay que desechar las columnas M y filas R que son irrelevantes para
    #el usuario ii, para poder incrementar la velocidaddel algorítmo

    M_selected <- M[, W[ii,] == 1, drop = FALSE]
    R_selected <- R[ii,][W[ii,] == 1, drop = FALSE]
```

```
    # Actualiza U para cada usuario ii
    U[ii,] <-
      t(solve(
        M_selected %*% t(M_selected) + lambda * n_u_i[ii] * diag(n_factors),
        (M_selected %*% R_selected)
      ))


  }
  cost <- fn_costo(R, U, M, W, lambda = lambda, n_u_i, n_m_j)
  print(paste0(kk, "iteración, paso 2: Función de costo = ", cost))
  vector_kkk[kk] <- cost

  # Minimizar U %*% M para una U fija iterando sobre n usuarios (items)
  for (jj in 1:n_m) {
    if (sum(W[, jj] == 1) > 0) {
      U_selected <- U[W[, jj] == 1, , drop = FALSE]
      R_col_selected <- R[, jj][W[, jj] == 1, drop = FALSE]
      # Actualiza M para cada película jj
      M[, jj] <-
        solve(
          t(U_selected) %*% U_selected + lambda * n_m_j[jj] * diag(n_factors),
          t(U_selected) %*% R_col_selected
        )
    }
  }
  cost <- fn_costo(R, U, M, W, lambda = lambda, n_u_i, n_m_j)
  print(paste0(kk, "iteración, paso 2: Función de costo = ", cost))
  vector_kkkk[kk] <- cost

  # Guardar iteración
  vector_kk[kk] <- kk

}
```

```
## [1] "1iteración, paso 2: Función de costo = 1747.30690497421"
## [1] "1iteración, paso 2: Función de costo = 1707.18003632765"
## [1] "2iteración, paso 2: Función de costo = 1672.28272330158"
## [1] "2iteración, paso 2: Función de costo = 1645.77735729699"
## [1] "3iteración, paso 2: Función de costo = 1610.34309548809"
## [1] "3iteración, paso 2: Función de costo = 1371.51133986819"
## [1] "4iteración, paso 2: Función de costo = 1044.19554586752"
## [1] "4iteración, paso 2: Función de costo = 1007.16295373624"
## [1] "5iteración, paso 2: Función de costo = 984.36347341824"
## [1] "5iteración, paso 2: Función de costo = 968.233566690092"
## [1] "6iteración, paso 2: Función de costo = 955.681920022508"
## [1] "6iteración, paso 2: Función de costo = 945.909190329845"
## [1] "7iteración, paso 2: Función de costo = 937.707763331105"
## [1] "7iteración, paso 2: Función de costo = 930.989665768773"
## [1] "8iteración, paso 2: Función de costo = 925.191441233792"
## [1] "8iteración, paso 2: Función de costo = 920.346048724563"
## [1] "9iteración, paso 2: Función de costo = 916.133030449553"
## [1] "9iteración, paso 2: Función de costo = 912.58688910472"
```

```
## [1] "10iteración, paso 2: Función de costo = 909.502037885307"
## [1] "10iteración, paso 2: Función de costo = 906.899251436826"
## [1] "11iteración, paso 2: Función de costo = 904.638001670788"
## [1] "11iteración, paso 2: Función de costo = 902.727216031235"
## [1] "12iteración, paso 2: Función de costo = 901.068236562954"
## [1] "12iteración, paso 2: Función de costo = 899.6629884289"
## [1] "13iteración, paso 2: Función de costo = 898.442315884698"
## [1] "13iteración, paso 2: Función de costo = 897.404989675921"
## [1] "14iteración, paso 2: Función de costo = 896.502957636707"
## [1] "14iteración, paso 2: Función de costo = 895.733798171405"
## [1] "15iteración, paso 2: Función de costo = 895.064249569301"
## [1] "15iteración, paso 2: Función de costo = 894.491537389295"
## [1] "16iteración, paso 2: Función de costo = 893.992593066978"
## [1] "16iteración, paso 2: Función de costo = 893.564659438879"
## [1] "17iteración, paso 2: Función de costo = 893.191653198121"
## [1] "17iteración, paso 2: Función de costo = 892.871013712836"
## [1] "18iteración, paso 2: Función de costo = 892.591454007007"
## [1] "18iteración, paso 2: Función de costo = 892.350695693629"
## [1] "19iteración, paso 2: Función de costo = 892.140763824398"
## [1] "19iteración, paso 2: Función de costo = 891.959692913721"
## [1] "20iteración, paso 2: Función de costo = 891.801811206824"
## [1] "20iteración, paso 2: Función de costo = 891.665462797161"
## [1] "21iteración, paso 2: Función de costo = 891.546589253591"
## [1] "21iteración, paso 2: Función de costo = 891.443821193452"
## [1] "22iteración, paso 2: Función de costo = 891.354238232224"
## [1] "22iteración, paso 2: Función de costo = 891.276724636658"
## [1] "23iteración, paso 2: Función de costo = 891.209168112518"
## [1] "23iteración, paso 2: Función de costo = 891.150670450791"
## [1] "24iteración, paso 2: Función de costo = 891.099696670997"
## [1] "24iteración, paso 2: Función de costo = 891.055530739735"
## [1] "25iteración, paso 2: Función de costo = 891.01705244609"
## [1] "25iteración, paso 2: Función de costo = 890.983695575519"
## [1] "26iteración, paso 2: Función de costo = 890.954639449591"
## [1] "26iteración, paso 2: Función de costo = 890.929439313929"
## [1] "27iteración, paso 2: Función de costo = 890.90749182692"
## [1] "27iteración, paso 2: Función de costo = 890.888449585595"
## [1] "28iteración, paso 2: Función de costo = 890.871867627139"
## [1] "28iteración, paso 2: Función de costo = 890.857475858735"
## [1] "29iteración, paso 2: Función de costo = 890.844945186954"
## [1] "29iteración, paso 2: Función de costo = 890.834066451126"
## [1] "30iteración, paso 2: Función de costo = 890.824595631082"
## [1] "30iteración, paso 2: Función de costo = 890.816371291091"
## [1] "31iteración, paso 2: Función de costo = 890.809212076084"
## [1] "31iteración, paso 2: Función de costo = 890.802993737263"
## [1] "32iteración, paso 2: Función de costo = 890.7975812144"
## [1] "32iteración, paso 2: Función de costo = 890.792879108983"
## [1] "33iteración, paso 2: Función de costo = 890.788786652529"
## [1] "33iteración, paso 2: Función de costo = 890.785230747581"
## [1] "34iteración, paso 2: Función de costo = 890.782136087059"
## [1] "34iteración, paso 2: Función de costo = 890.779446759045"
## [1] "35iteración, paso 2: Función de costo = 890.777106401944"
## [1] "35iteración, paso 2: Función de costo = 890.775072312789"
## [1] "36iteración, paso 2: Función de costo = 890.77330225417"
## [1] "36iteración, paso 2: Función de costo = 890.771763653162"
```

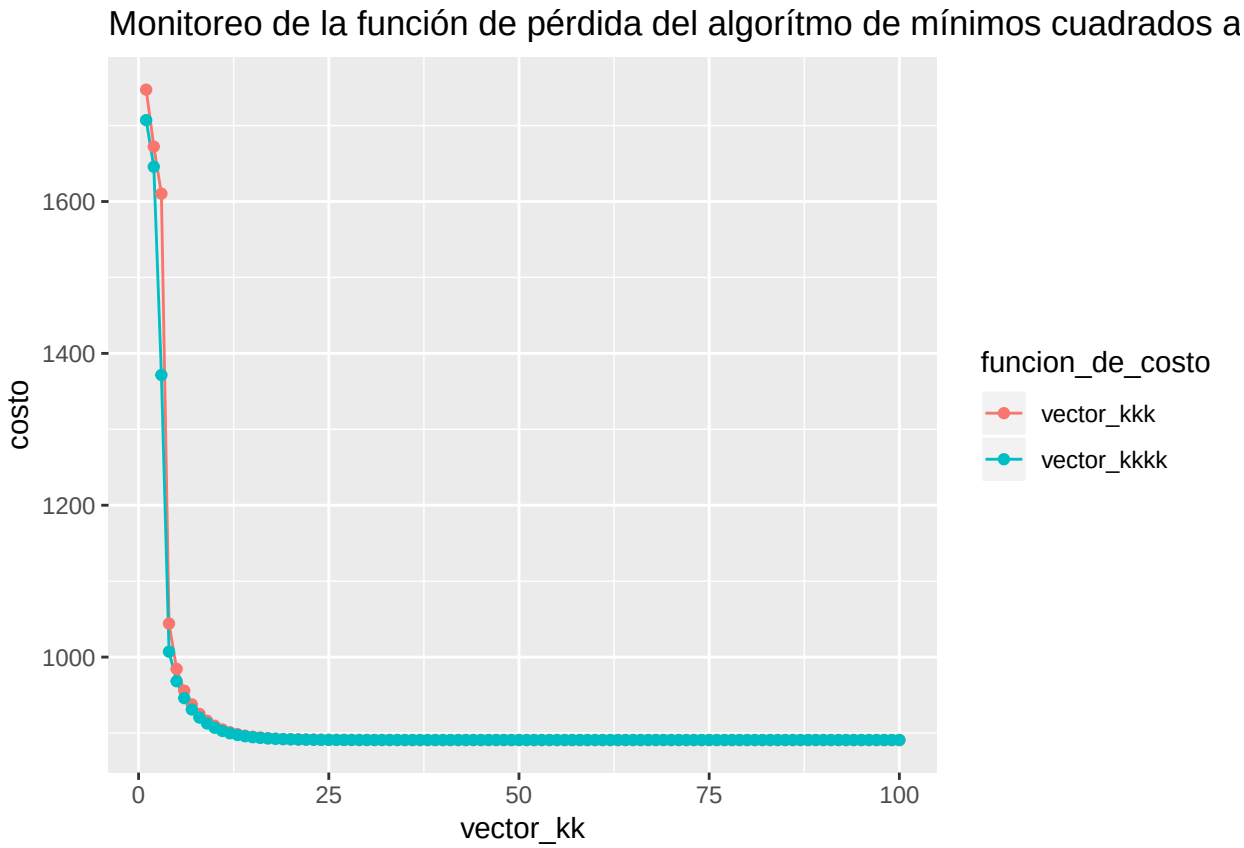
```
## [1] "37iteración, paso 2: Función de costo = 890.770424820302"
## [1] "37iteración, paso 2: Función de costo = 890.769260936832"
## [1] "38iteración, paso 2: Función de costo = 890.768248202595"
## [1] "38iteración, paso 2: Función de costo = 890.76736772482"
## [1] "39iteración, paso 2: Función de costo = 890.766601612222"
## [1] "39iteración, paso 2: Función de costo = 890.765935494635"
## [1] "40iteración, paso 2: Función de costo = 890.765355911753"
## [1] "40iteración, paso 2: Función de costo = 890.764851940926"
## [1] "41iteración, paso 2: Función de costo = 890.764413448178"
## [1] "41iteración, paso 2: Función de costo = 890.764032136208"
## [1] "42iteración, paso 2: Función de costo = 890.763700370514"
## [1] "42iteración, paso 2: Función de costo = 890.763411851322"
## [1] "43iteración, paso 2: Función de costo = 890.763160823895"
## [1] "43iteración, paso 2: Función de costo = 890.762942507176"
## [1] "44iteración, paso 2: Función de costo = 890.762752561201"
## [1] "44iteración, paso 2: Función de costo = 890.762587358861"
## [1] "45iteración, paso 2: Función de costo = 890.762443625712"
## [1] "45iteración, paso 2: Función de costo = 890.762318610958"
## [1] "46iteración, paso 2: Función de costo = 890.762209843113"
## [1] "46iteración, paso 2: Función de costo = 890.762115236548"
## [1] "47iteración, paso 2: Función de costo = 890.762032925193"
## [1] "47iteración, paso 2: Función de costo = 890.761961328113"
## [1] "48iteración, paso 2: Función de costo = 890.761899035921"
## [1] "48iteración, paso 2: Función de costo = 890.761844850488"
## [1] "49iteración, paso 2: Función de costo = 890.761797707058"
## [1] "49iteración, paso 2: Función de costo = 890.76175669777"
## [1] "50iteración, paso 2: Función de costo = 890.76172101804"
## [1] "50iteración, paso 2: Función de costo = 890.761689980049"
## [1] "51iteración, paso 2: Función de costo = 890.761662975682"
## [1] "51iteración, paso 2: Función de costo = 890.761639483898"
## [1] "52iteración, paso 2: Función de costo = 890.761619044989"
## [1] "52iteración, paso 2: Función de costo = 890.761601264292"
## [1] "53iteración, paso 2: Función de costo = 890.761585794232"
## [1] "53iteración, paso 2: Función de costo = 890.761572335896"
## [1] "54iteración, paso 2: Función de costo = 890.761560626452"
## [1] "54iteración, paso 2: Función de costo = 890.761550439527"
## [1] "55iteración, paso 2: Función de costo = 890.761541576341"
## [1] "55iteración, paso 2: Función de costo = 890.761533865466"
## [1] "56iteración, paso 2: Función de costo = 890.761527156551"
## [1] "56iteración, paso 2: Función de costo = 890.761521319785"
## [1] "57iteración, paso 2: Función de costo = 890.761516241432"
## [1] "57iteración, paso 2: Función de costo = 890.761511823197"
## [1] "58iteración, paso 2: Función de costo = 890.761507979038"
## [1] "58iteración, paso 2: Función de costo = 890.761504634527"
## [1] "59iteración, paso 2: Función de costo = 890.761501724567"
## [1] "59iteración, paso 2: Función de costo = 890.761499192803"
## [1] "60iteración, paso 2: Función de costo = 890.761496989979"
## [1] "60iteración, paso 2: Función de costo = 890.761495073428"
## [1] "61iteración, paso 2: Función de costo = 890.761493405878"
## [1] "61iteración, paso 2: Función de costo = 890.761491955024"
## [1] "62iteración, paso 2: Función de costo = 890.761490692661"
## [1] "62iteración, paso 2: Función de costo = 890.761489594331"
## [1] "63iteración, paso 2: Función de costo = 890.761488638688"
## [1] "63iteración, paso 2: Función de costo = 890.761487807216"
```

```
## [1] "64iteración, paso 2: Función de costo = 890.761487083761"
## [1] "64iteración, paso 2: Función de costo = 890.761486454301"
## [1] "65iteración, paso 2: Función de costo = 890.761485906613"
## [1] "65iteración, paso 2: Función de costo = 890.76148543008"
## [1] "66iteración, paso 2: Función de costo = 890.761485015451"
## [1] "66iteración, paso 2: Función de costo = 890.761484654688"
## [1] "67iteración, paso 2: Función de costo = 890.761484340788"
## [1] "67iteración, paso 2: Función de costo = 890.761484067666"
## [1] "68iteración, paso 2: Función de costo = 890.761483830022"
## [1] "68iteración, paso 2: Función de costo = 890.761483623249"
## [1] "69iteración, paso 2: Función de costo = 890.761483443334"
## [1] "69iteración, paso 2: Función de costo = 890.76148328679"
## [1] "70iteración, paso 2: Función de costo = 890.761483150579"
## [1] "70iteración, paso 2: Función de costo = 890.761483032061"
## [1] "71iteración, paso 2: Función de costo = 890.761482928938"
## [1] "71iteración, paso 2: Función de costo = 890.761482839209"
## [1] "72iteración, paso 2: Función de costo = 890.761482761134"
## [1] "72iteración, paso 2: Función de costo = 890.7614826932"
## [1] "73iteración, paso 2: Función de costo = 890.76148263409"
## [1] "73iteración, paso 2: Función de costo = 890.761482582657"
## [1] "74iteración, paso 2: Función de costo = 890.761482537904"
## [1] "74iteración, paso 2: Función de costo = 890.761482498963"
## [1] "75iteración, paso 2: Función de costo = 890.76148246508"
## [1] "75iteración, paso 2: Función de costo = 890.761482435598"
## [1] "76iteración, paso 2: Función de costo = 890.761482409944"
## [1] "76iteración, paso 2: Función de costo = 890.761482387622"
## [1] "77iteración, paso 2: Función de costo = 890.7614823682"
## [1] "77iteración, paso 2: Función de costo = 890.761482351299"
## [1] "78iteración, paso 2: Función de costo = 890.761482336594"
## [1] "78iteración, paso 2: Función de costo = 890.761482323798"
## [1] "79iteración, paso 2: Función de costo = 890.761482312664"
## [1] "79iteración, paso 2: Función de costo = 890.761482302976"
## [1] "80iteración, paso 2: Función de costo = 890.761482294545"
## [1] "80iteración, paso 2: Función de costo = 890.76148228721"
## [1] "81iteración, paso 2: Función de costo = 890.761482280828"
## [1] "81iteración, paso 2: Función de costo = 890.761482275274"
## [1] "82iteración, paso 2: Función de costo = 890.761482270441"
## [1] "82iteración, paso 2: Función de costo = 890.761482266236"
## [1] "83iteración, paso 2: Función de costo = 890.761482262577"
## [1] "83iteración, paso 2: Función de costo = 890.761482259393"
## [1] "84iteración, paso 2: Función de costo = 890.761482256622"
## [1] "84iteración, paso 2: Función de costo = 890.761482254212"
## [1] "85iteración, paso 2: Función de costo = 890.761482252114"
## [1] "85iteración, paso 2: Función de costo = 890.761482250289"
## [1] "86iteración, paso 2: Función de costo = 890.761482248701"
## [1] "86iteración, paso 2: Función de costo = 890.761482247319"
## [1] "87iteración, paso 2: Función de costo = 890.761482246116"
## [1] "87iteración, paso 2: Función de costo = 890.76148224507"
## [1] "88iteración, paso 2: Función de costo = 890.761482244159"
## [1] "88iteración, paso 2: Función de costo = 890.761482243367"
## [1] "89iteración, paso 2: Función de costo = 890.761482242677"
## [1] "89iteración, paso 2: Función de costo = 890.761482242077"
## [1] "90iteración, paso 2: Función de costo = 890.761482241555"
## [1] "90iteración, paso 2: Función de costo = 890.761482241101"
```

```
## [1] "91iteración, paso 2: Función de costo = 890.761482240706"
## [1] "91iteración, paso 2: Función de costo = 890.761482240362"
## [1] "92iteración, paso 2: Función de costo = 890.761482240063"
## [1] "92iteración, paso 2: Función de costo = 890.761482239802"
## [1] "93iteración, paso 2: Función de costo = 890.761482239576"
## [1] "93iteración, paso 2: Función de costo = 890.761482239378"
## [1] "94iteración, paso 2: Función de costo = 890.761482239207"
## [1] "94iteración, paso 2: Función de costo = 890.761482239058"
## [1] "95iteración, paso 2: Función de costo = 890.761482238928"
## [1] "95iteración, paso 2: Función de costo = 890.761482238814"
## [1] "96iteración, paso 2: Función de costo = 890.761482238716"
## [1] "96iteración, paso 2: Función de costo = 890.76148223863"
## [1] "97iteración, paso 2: Función de costo = 890.761482238556"
## [1] "97iteración, paso 2: Función de costo = 890.761482238491"
## [1] "98iteración, paso 2: Función de costo = 890.761482238435"
## [1] "98iteración, paso 2: Función de costo = 890.761482238386"
## [1] "99iteración, paso 2: Función de costo = 890.761482238343"
## [1] "99iteración, paso 2: Función de costo = 890.761482238306"
## [1] "100iteración, paso 2: Función de costo = 890.761482238273"
## [1] "100iteración, paso 2: Función de costo = 890.761482238245"
```

```
#print(vector_kk)
#print(vector_kkk)
#print(vector_kkkk)
data_frame(vector_kk, vector_kkk,vector_kkkk) %>%
  gather("funcion_de_costo","costo",2:3) %>% ggplot(aes(x = vector_kk, y = costo, color = funcion_de_cos
  geom_line() +
  geom_point() +
  ggtitle("Monitoreo de la función de pérdida del algorítmo de mínimos cuadrados alternados")
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```



Recuperamos la matriz U que corresponde a usuarios y k factores latentes. En teoría esperaríamos que esta descomposición se pudiera interpretar como géneros de películas que le gustan a los usuarios, pero vemos que no es tán fácil identificar esta información a partir de la matriz.

Abajo se muestran las 10 primeras entradas de la matriz U estiamda.

```
# Recuperamos la matriz U que corresponde a usuarios y k factores latentes
head(round(U,1),10)
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]  2.2 -0.1 -0.4 -0.4  0.2
##  [2,]  1.9  0.6 -0.4 -0.5 -0.1
##  [3,]  1.9  0.2  1.0  0.9 -0.2
##  [4,]  1.8 -0.2 -0.9 -1.1 -0.2
##  [5,]  1.3 -0.2 -1.2 -0.3 -0.6
##  [6,]  1.9 -0.6  1.2  0.3  0.0
##  [7,]  1.5  0.1  0.0 -1.7  0.2
##  [8,]  1.7  1.3 -1.1 -0.5  0.7
##  [9,]  2.1 -0.1 -0.8  1.1  0.5
## [10,]  1.5 -1.5 -0.5  0.4  0.2
```

Está matriz se debería interpretar como los géneros a los que pertenece cada una de las películas, pero igualmente no es tan fácil interpretar a partir de la matriz.

Abajo se muestran las 10 primeras entradas de la matriz M estiamda.

```r
# Recuperamos la matriz M que corresponde a películas y k factores latentes
round(M,1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  2.9  1.8  1.7  1.5  1.8  1.8  1.8  1.3  1.6   1.6
## [2,]  0.0 -0.6 -0.6  1.4 -0.1  0.2  0.4  0.4 -0.3  -1.3
## [3,]  0.0  1.0 -0.5 -0.5 -0.7  0.8 -0.4  1.2  0.2  -0.9
## [4,]  0.0  0.7 -0.4  0.4  1.0 -0.9 -0.9  0.9 -0.6   0.3
## [5,]  0.0  0.8 -0.7  0.0 -0.4 -1.0  0.8 -0.5  1.4  -0.1
```

Finalmente, calculamos el error RMSE

```r
# Aproximamos la matriz de calificaciones R realizando el producto de matrices U * M
ratings <-as(U %*% M, "dgCMatrix")
round(ratings)
```

```
## 100 x 10 sparse Matrix of class "dgCMatrix"
##
##   [1,] 7 4 4  3  4 4 5  2  4  4
##   [2,] 6 2 3  4  3 4 4  2  3  2
##   [3,] 5 5 2  3  4 4 2  5  3  2
##   [4,] 5 1 4  2  3 4 4  0  3  4
##   [5,] 4 0 3  2  3 2 3  0  1  3
##   [6,] 6 5 3  2  3 4 2  4  3  3
##   [7,] 4 2 3  2  1 4 5  0  4  2
##   [8,] 5 1 2  5  3 2 5  1  3  2
##   [9,] 6 4 3  4  5 2 4  2  3  5
##  [10,] 5 3 3  1  3 2 2  1  3  5
##  [11,] 6 2 3  5  5 3 4  3  2  3
##  [12,] 5 2 4  2  3 4 2  3  1  3
##  [13,] 6 2 5  4  4 4 4  1  3  4
##  [14,] 4 2 1  4  1 5 4  4  2 -1
##  [15,] 4 3 2  1  3 0 3  0  3  4
##  [16,] 4 5 1  1  2 3 1  4  2  1
##  [17,] 4 1 3  4  3 4 2  3  0  1
##  [18,] 2 3 1  1  1 1 1  2  2  1
##  [19,] 5 2 4  3  3 5 3  3  2  2
##  [20,] 3 4 1 -1  0 2 2  2  4  2
##  [21,] 7 6 3  3  3 5 4  4  5  2
##  [22,] 4 4 2  1  2 3 1  3  2  2
##  [23,] 3 2 1  1  1 2 2  1  2  1
##  [24,] 4 3 2  2  2 2 4  1  4  2
##  [25,] 5 2 1  5  1 4 4  3  2 -2
##  [26,] 3 1 3  1  2 2 3  0  2  3
##  [27,] 3 1 1  4  1 4 2  3  1 -1
##  [28,] 5 2 4  3  5 4 2  3  0  4
##  [29,] 5 4 4  0  3 3 4  1  5  5
##  [30,] 6 4 3  3  5 3 2  4  2  3
##  [31,] 5 3 3  3  3 2 4  2  4  3
##  [32,] 5 4 2  2  2 3 4  2  5  2
##  [33,] 4 4 1  2  2 3 3  3  4  1
##  [34,] 4 1 3  1  2 2 4  0  3  3
```

```
##  [35,] 4 4 2  1  1 4 2  3  3  1
##  [36,] 4 3 2  3  3 2 2  3  1  2
##  [37,] 5 2 2  4  3 2 4  1  3  2
##  [38,] 5 2 4  2  3 4 3  1  2  3
##  [39,] 6 4 3  4  4 2 4  3  4  3
##  [40,] 4 2 2  2  2 3 3  2  2  1
##  [41,] 6 4 4  4  5 4 2  4  1  3
##  [42,] 5 2 4  1  4 3 2  1  1  5
##  [43,] 5 4 3  1  2 4 4  2  4  3
##  [44,] 3 2 1  1 -1 5 4  2  4 -1
##  [45,] 3 1 1  2  0 3 4  1  3  0
##  [46,] 4 5 2  1  3 0 3  1  5  4
##  [47,] 5 5 2  3  4 1 3  3  4  3
##  [48,] 5 2 3  2  3 3 3  1  3  3
##  [49,] 2 1 2  2  4 1 0  2 -1  2
##  [50,] 4 3 2  3  3 2 2  3  2  2
##  [51,] 4 3 3  0  2 4 1  3  2  3
##  [52,] 5 4 2  2  2 1 5  1  5  3
##  [53,] 3 2 2  1  1 4 1  3  1  1
##  [54,] 6 3 3  4  4 3 4  3  3  3
##  [55,] 4 2 3  1  2 5 2  3  2  1
##  [56,] 4 2 3  2  3 2 1  2  1  3
##  [57,] 4 1 3  4  4 2 3  2  1  2
##  [58,] 6 4 2  3  2 4 4  3  4  1
##  [59,] 5 2 3  4  5 2 2  3  0  3
##  [60,] 5 4 4  1  3 4 2  3  2  4
##  [61,] 4 5 2  0  2 1 4  1  5  4
##  [62,] 5 2 4  2  3 3 3  2  2  3
##  [63,] 4 2 2  3  2 1 4  0  4  2
##  [64,] 6 5 2  3  3 3 3  4  4  2
##  [65,] 6 4 4  3  4 6 2  5  1  2
##  [66,] 5 1 4  2  2 4 5  0  3  4
##  [67,] 4 2 2  4  4 3 1  4  0  2
##  [68,] 5 2 4  2  2 5 2  3  2  2
##  [69,] 5 4 3  0  3 2 2  1  3  5
##  [70,] 4 2 2  2  3 2 2  2  2  3
##  [71,] 5 5 2  3  3 3 3  4  4  2
##  [72,] 5 2 3  2  2 4 3  2  2  2
##  [73,] 7 4 4  3  4 3 5  1  5  5
##  [74,] 5 5 2  3  5 1 1  4  2  3
##  [75,] 5 2 3  4  5 2 3  2  2  3
##  [76,] 6 2 4  4  3 5 5  1  3  3
##  [77,] 5 3 2  3  1 3 5  2  5  1
##  [78,] 6 2 3  4  4 3 4  2  3  3
##  [79,] 6 2 5  3  4 2 6 -1  4  6
##  [80,] 5 5 3  1  3 4 2  4  3  3
##  [81,] 7 3 5  3  4 5 4  2  3  4
##  [82,] 6 2 5  2  4 3 3  1  2  6
##  [83,] 5 2 4  1  2 3 5 -1  4  4
##  [84,] 5 5 2  2  2 3 4  2  5  2
##  [85,] 4 0 2  4  2 4 3  2  1  0
##  [86,] 4 4 2  1  2 2 2  2  3  2
##  [87,] 4 1 3  2  1 4 3  1  2  1
##  [88,] 7 3 5  4  5 4 4  3  2  4
```

```
##  [89,] 5 1 2  4  3 2 4  1  3  2
##  [90,] 5 4 3  2  3 3 4  2  4  3
##  [91,] 6 2 4  3  2 6 5  2  4  2
##  [92,] 6 5 4  1  4 3 3  3  4  5
##  [93,] 5 4 2  3  4 1 4  2  4  3
##  [94,] 5 3 3  1  3 4 2  3  2  2
##  [95,] 4 2 3  3  5 0 2  1  1  4
##  [96,] 5 4 3  2  3 3 3  3  3  3
##  [97,] 5 4 2  3  4 1 3  2  3  4
##  [98,] 5 4 2  3  2 3 4  2  4  1
##  [99,] 6 4 4  2  4 4 3  3  3  4
## [100,] 5 2 4  2  4 4 2  3  1  3
```

Calculamos el error de la aproximación utilizando RMSE.

```
# Error de la aproximación RMSE
recm(R,ratings)
```

```
## [1] 2.179119
```

Vemos que el error es grande para el contexto del problema, pero luce bien para la implementación de este algorítmo.

El siguiente paso sería ver como se ve está implementación utilizando todos los datos y paralelización, utilizando el algorítmo ALS programado en Spark.

ALS paralelizado con todos los datos de netflix.

Referencias

Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering

https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix

Large-scale Parallel Collaborative Filtering for the Netflix Prize

https://endymecy.gitbooks.io/spark-ml-source-analysis/content/%E6%8E%A8%E8%8D%90/papers/Large-scale%20Parallel%20Collaborative%20Filtering%20the%20Netflix%20Prize.pdf

Matrix Completion via Alternating Least Square(ALS)

http://stanford.edu/~rezab/classes/cme323/S15/notes/lec14.pdf

Alternating Least Squares Method for Collaborative Filtering

https://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-method-for-collaborative-filtering/