# 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



# 1 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 personas con gustos similares. Usamos esas predicciones para hacer recomendaciones personalizadas de películas basadas en los gustos únicos de cada cliente.

## 2 Objetivo:

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

## 3 Mínimos cuadrados alternados

El propóstio de un sistema de recomendación es sugerir productos que en caso de consumirse, maximizan la utilidad del comprador y los beneficios del vendedor en un determinado momento en el tiempo. Estos sistemas analizan y procesan la información histórica de los usuarios, como por ejemplo, compras previas, califiaciones o visitas. Asismimo, también utilizan información histórica de los productos, como por ejemplo, marcas, titulos, precios, etc. Esta información se combina para predecir qué producto puede ser interesante para el consumidor.

Los sistemas de recomendación pueden ser de dos tipos:

- filtros colaborativos
- filtros basados en contenido

El método de mínimos cuadarado alterneados es un filtro colaborativo que matematicamente es un problema de factorización de matrices. El propóstito de este algorítmo es resolver un problema de optimización para encontrar las matrices $U$ y $M$ que multiplicadas puedan aproximar la matriz de datos original $R$. Dicho de otra manera el problema se ve de la siguiente manera $R \approx UM$.

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

$$R \approx UM^t$$

Ahora supongamos que conocemos $M_1$. Si este es el caso, entonces queremos resolver para $U_1$:

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

Como $M_1^t$ 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 $M$:

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

Y una vez que encontramos $M_2$ 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.

# 4 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 $U \times M$, 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 recibe 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 qué tanto le gusta al usuario ese género de películas (factores latentes).

# 5 Mínimos cuadrados alternados con regularización

En el caso de los problemas de sistemas de recomendación las matrices $R$ tienen la característica de ser muy ralas porque no todos los usuarios han revisado todos los productos. En el contexto del problema de las películas de Netflix, en promedio los usuarios han calificado 209 de 17,770 películas. Para lidiar lidiar con estos datos ralos implementaremos la solución de minimos cuadrados alternados con **regularización** (Zohu, Y. et al, 2008).

El problema de optimización se reescribe de la siguiente manera:

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

El primer término a la derecha minimiza $R{-}U{\times}M$ 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 el rating. De este modo $W$ garantiza que solo se tomen 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 $m_j$

En el paper de Zohu et al, 2008 describen la resolución matemática para obtener la matriz $U$ cuando se conoce $M$ y $M$ cuando se conoce $U$ que se describe a continuación.

En el caso de la resolución de la matriz $U$ cuando se conoce $M$:

Una fila dada de $U$, por ejemplo $u_i$, se determina resolviendo un problema lineal de mínimos cuadrados que involucra los ratings conocidos del usuario $i$, y los vectores de características $m_j$ de las películas que el

usuario $i$ ha calificado. Dicho de otra manera cada fila de usuarios será un problema de mínimos cuadrados regularizados.

Abajo se muestra la derivación del problema de optimización.

$$\frac{1}{2}\frac{\partial f}{\partial u_{ki}} = 0, \forall i, k$$

$$\Rightarrow \sum_{j \in I_i} (\mathbf{u}_i^T \mathbf{m}_j - r_{ij}) m_{kj} + \lambda n_{u_i} u_{ki} = 0, \forall i, k$$

$$\Rightarrow \sum_{j \in I_i} m_{kj} \mathbf{m}_j^T \mathbf{u}_i + \lambda n_{u_i} u_{ki} = \sum_{j \in I_i} m_{kj} r_{ij}, \forall i, k$$

$$\Rightarrow \left(M_{I_i} M_{I_i}^T + \lambda n_{u_i} E\right) \mathbf{u}_i = M_{I_i}, R^T(i, I_i), \forall i$$

$$\Rightarrow \mathbf{u}_i = A_i^{-1} V_i, \forall i$$

donde $A = M_{I_i} M_{I_i}^t + \lambda n_{u_i} E$, $V_i = M_{I_i} R^T(i, I_i)$ $E$ es la matriz identidad $n_f \times n_f$. $M_{I_i}$ es la submatriz de $M$ que contiene las películas que el usuario $i$ ha calificado. $R(i, I_i)$ es un vector fila donde las columnas son las películas que el usuario ha calificado.

Similarmente, la matriz $M$ se actualizará columna a columna resolviendo un problema de mínimos cuadrados con regularización. Siguiendo la deriviación para la matriz $U$, la matriz $M$ se resuelve como se muestra abajo.

$$\mathbf{m}_j = A_j^{-1} V_j, \forall_j$$

donde $A = U_{I_j} U_{I_j}^t + \lambda n_{m_j} E$, $V_i = U_{I_j} R(I_j, j)$ . $U_{I_j}$ es la submatriz de $U$ que contiene únicamente los usuarios que han visto la película $j$.

# 6 Algoritmo

Con lo mencionado anteriormente el algoritmo queda:

1. Se inicializa la matriz $M$ asignando la calificación promedio para esa película como primera fila y pequeños números aleatorios para las entradas restantes.
2. Se fijar $M$ y para cada renglón de $U$ (cada usuario) se resuelve el problema de mínimos cuadrado con:
$$\mathbf{u}_j = A_i^{-1} V_i$$
3. Se fija $U$ y para cada columna de $M$ (cada pelicula) se resuelve el problema de mínimos cuadrado con:
$$\mathbf{m}_j = A_j^{-1} V_j$$

# 7 Implementación del algoritmo

## 7.1 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:

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

Adicionalmente, se proporcionó un catálogo de películas.

En el trabajo utilizaremos una muestra pequeña de los datos de Netflix que representa el 1% de los usuarios y 1% de las películas. En total la matriz de datos contiene 129 número de películas, 769 número de usuarios y equivale a 99,201 entradas. Esto en razón de que el tamaño del conjunto de datos es muy grande, muy ralo y el costo computacional para un implementación en una máquina local es altísimo. Recordemos que el objetivo del proyecto es implementar el algorítmo y demostrar que funciona de manera correcta.

## 7.2 Algoritmo implementado

```
functionALS <- function(datos,             # dataframe con los datos películas-usuarios
                        lambda = 0.03,     # parámetro de regularización
                        n_factores = 5,    # número de variables latentes
                        n_iter = 100,      # número de iteraciones
                        delta = 0.0001)    # parámetro para escalar los datos
  {

  # Semilla
  set.seed(28882)

  # Comenzamos a contar el tiempo de ejecución del algoritmo
  start_time <- Sys.time()
  # Convertimos los datos a una matrix de formato: usuarios = X películas = R
  R <- sparseMatrix(
    as.integer(datos$nuevos_usuarios_ids), # renglón usuarios
    as.integer(datos$nuevos_ids),          # columnas de películas
    x = datos$calif)                       # calificaciones

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

  # 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 que sí calificó (a esta matriz le llaman I en el paper)
```

```r
W <- R
W@x[!is.na(W@x)] <- 1

######  Definimos función de costo   ######

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))
}

######  Inicializamos el algoritmo   ######

# 1. Inicializamos la matriz de películas y factores latentes
M <- delta * matrix(runif(n_m * n_factores),nrow = n_factores, ncol = n_m)

# Como se sugiere en el artículo, se inicializará la primera fila de la matriz M
# con los promedios de calificaciones de las películas. Si hay películas SIN
# calificación, se les pone la calificación promedio de todas las peliculas
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_factores)
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)


###### EJECUCION DEL ALGORITMO #######

# Inicializamos vectores para guardar el monitoreo
iteracion <- vector(mode = "numeric", length = n_iter)
costo1 <- vector(mode = "numeric", length = n_iter)
costo2 <- vector(mode = "numeric", length = n_iter)


# Este for aproxima U%*%M a R, considerando la matriz de pesos W para calcular el error
for (kk in 1:n_iter) {
  # 2. Se fija M y se minimiza 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 velocidad del algorítmo

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

    A<- M_selected %*% t(M_selected) + lambda * n_u_i[ii] * diag(n_factores)
```

```
    V <-  (M_selected %*% R_selected)

    # Actualiza U para cada usuario ii
    U[ii,] <- t(solve(A)%*%V)
  }

  # Calculas la función de costo para este paso
  cost <- fn_costo(R, U, M, W, lambda = lambda, n_u_i, n_m_j)
  print(paste0(kk, " iteración, paso 1: Función de costo = ", cost))
  costo1[kk] <- cost

  #  3. Minimizar U %*% M para una U fija iterando sobre n películas
  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]

      A <- t(U_selected) %*% U_selected + lambda * n_m_j[jj] * diag(n_factores)
      V <- t(U_selected) %*% R_col_selected
      # Actualiza M para cada película jj
      M[, jj] <- solve(A)%*%V
    }
  }

  # Calculas la función de costo para este paso
  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))
  costo2[kk] <- cost

  # Guardar iteración
  iteracion[kk] <- kk

}
end_time <- Sys.time()

###### RESULTADO #######

# A continuación se indican los outputs de las función

# Dataframe con el monitoreo de las iteraciones
monitoreo <- data_frame(iteracion, costo1,costo2)

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

# Error de la aproximación RMSE
recm <- function(calif, pred){
  sqrt(mean((calif - pred)^2))
}
rmse <- recm(R,ratings)

# Tiempo de ejecución
tiempo <- end_time-start_time
```

```
  # Regresar todo esto como una lista
  resultados <- list(monitoreo = monitoreo,
                     rmse = rmse,
                     ratings = round(ratings),
                     tiempo = tiempo,
                     n_iter = n_iter,
                     lambda = lambda,
                     n_factores = n_factores,
                     delta = delta)
  return(resultados)
}
```

# 8 Evaluación del algorítmo

En esta sección se prueba el funcionamiento del algorítmo utilizando la muestra de datos del concurso de Netflix.

- Tiempo de ejecución del algorítmo
- Error cuadrático medio
- Curvas de la función de pérdida contra las iteraciones
- Comparar con diferentes configuraciones de lambda y número de factores latentes

Vemos que la regularización sí tiene un impacto en la minimización de la función de costos. Para estos datos el modelo con regularización chica tuvo mejor desempeño (Ver figura 1).

Para cada modelo con los mismos factores la mejor minimización se obtuvo con regularización igual a 0.001 (Ver figura 2).

En la medida que aumenta el número de factores, se incrementa el tiempo de ejecución del algorítmo (Ver figura 3). Asimismo, se puede observar que el efecto de la regularización puede ayudar a reducir el tiempo en algunos casos. El modelo que menos tiempo tardó fue el que tenía una regularización de 0.1 y 20 factores.

El error cuadrático medio bajó en la medida que se incluyeron más factores y la lambda de regularización se hizo más chica (Ver figura 4). El modelo que mejor minimizó el RMSE fue la configuración con 50 factores y regularización de 0.001 que obtuvo una RMSE de 13.57805.

La relación entre el error cuadrático medio y el tiempo nos dice que en la medida que se aumente el número de factores con baja regularización es posible minimizar el RMSE pero habrá un trade off con el tiempo de ejecución del algorítmo (Ver figura 5).

Table 1: Resultados para las distintas configuraciones

| modelo | lambda | factores | tiempo | costo1 | costo2 | rmse |
|---|---|---|---|---|---|---|
| 12 | 0.001 | 8 | 5.566361 | 29423.222 | 29419.508 | 41.79916 |
| 9 | 0.001 | 10 | 5.547596 | 8477.014 | 8472.895 | 25.31401 |
| 15 | 0.001 | 15 | 5.997478 | 3739.275 | 3737.059 | 14.54376 |
| 3 | 0.001 | 20 | 5.620160 | 3230.613 | 3228.805 | 14.38531 |
| 6 | 0.001 | 50 | 6.691347 | 2989.641 | 2987.709 | 13.57805 |
| 11 | 0.010 | 8 | 5.775080 | 33107.149 | 33100.874 | 21.09796 |
| 8 | 0.010 | 10 | 5.453610 | 14239.891 | 14235.059 | 16.78945 |
| 14 | 0.010 | 15 | 6.042544 | 7630.530 | 7627.904 | 14.75821 |
| 2 | 0.010 | 20 | 5.704400 | 7452.808 | 7450.291 | 14.71185 |
| 5 | 0.010 | 50 | 6.569537 | 7458.932 | 7456.271 | 14.84034 |

| modelo | lambda | factores | tiempo | costo1 | costo2 | rmse |
|---|---|---|---|---|---|---|
| 10 | 0.100 | 8 | 5.529866 | 85314.863 | 85314.847 | 18.16429 |
| 7 | 0.100 | 10 | 5.667921 | 65950.974 | 65950.960 | 17.61210 |
| 13 | 0.100 | 15 | 5.832239 | 57743.315 | 57743.296 | 16.96545 |
| 1 | 0.100 | 20 | 5.389794 | 57398.279 | 57398.271 | 16.90128 |
| 4 | 0.100 | 50 | 6.632453 | 57387.204 | 57387.198 | 16.89945 |

# 9 Predicciones

A continuación se muestran las predicciones que este algorítmo recomendaría para dos usuarios.

Table 2: Recomendaciones para el usuario id 10

| nombre | rating |
|---|---|
| Gladiator | 72 |
| Black Sheep | 72 |
| Resident Evil | 72 |
| The Devil's Advocate | 72 |
| Look Who's Talking | 78 |
| Keeping the Faith | 81 |
| Mallrats | 90 |
| Star Trek: Deep Space Nine: Season 2 | 90 |
| The Matrix | 90 |
| The Best of Friends: Vol. 2 | 97 |

Table 3: Recomendaciones para el usuario id 8

| nombre | rating |
|---|---|
| Clerks: Uncensored | 10 |
| Mallrats | 10 |
| Alien 3: Collector's Edition | 10 |
| Resident Evil | 10 |
| The Devil's Advocate | 10 |
| Kill Bill: Vol. 2 | 11 |
| GoldenEye | 11 |
| Pirates of the Caribbean: The Curse of the Black Pearl | 12 |
| Gladiator | 12 |
| The Matrix | 12 |

# 10 Implementación en paralelo

En esta sección se compara nuestra implementación contra la versión paralelizada y distribuida de spark. Dicha implementación se basa en el mismo paper.

```
library(sparklyr)

# conectar con "cluster" local
```

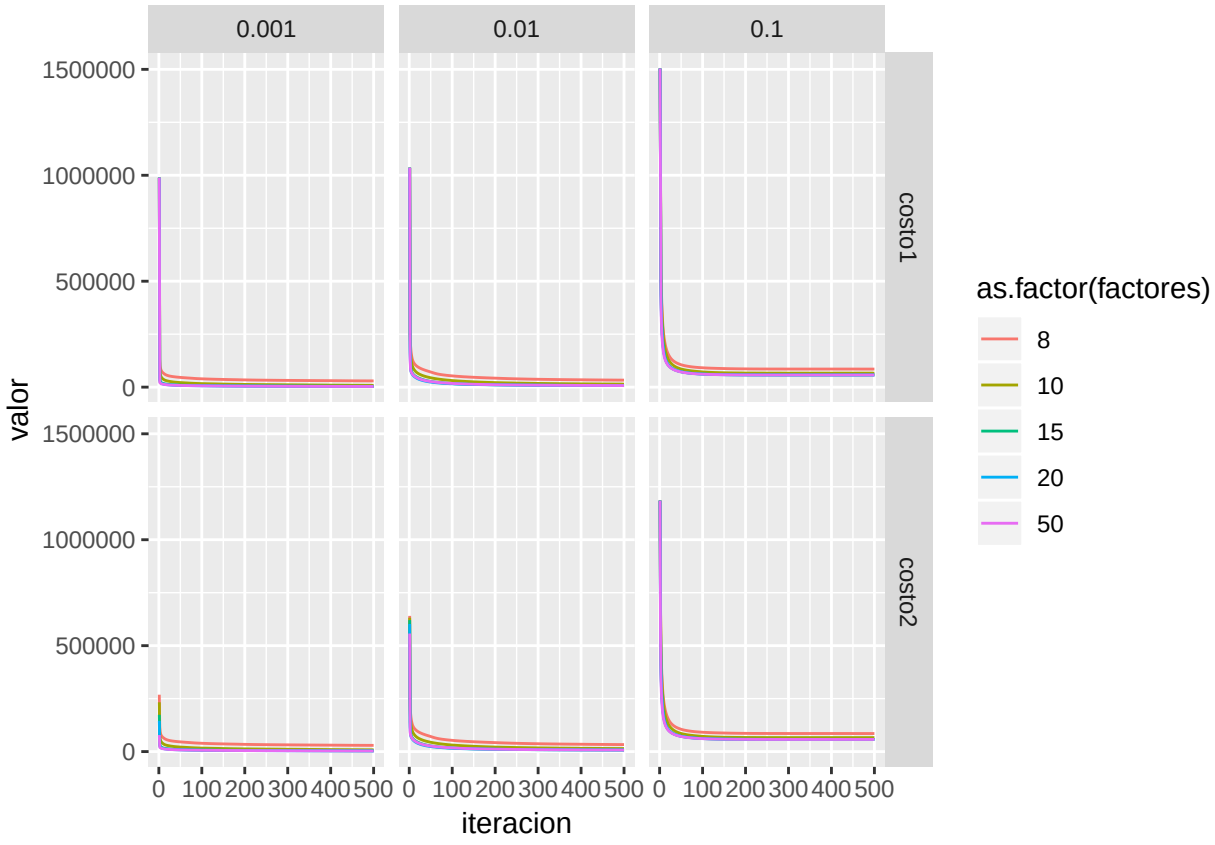


Figure 1: Relación entre la pérdida de la funciones de costos y el número de iteraciones del algoritmo de mínimos cuardados alternados. Los paneles están segementados por parámetro de regularización y los colores corresponden al númnero de factores

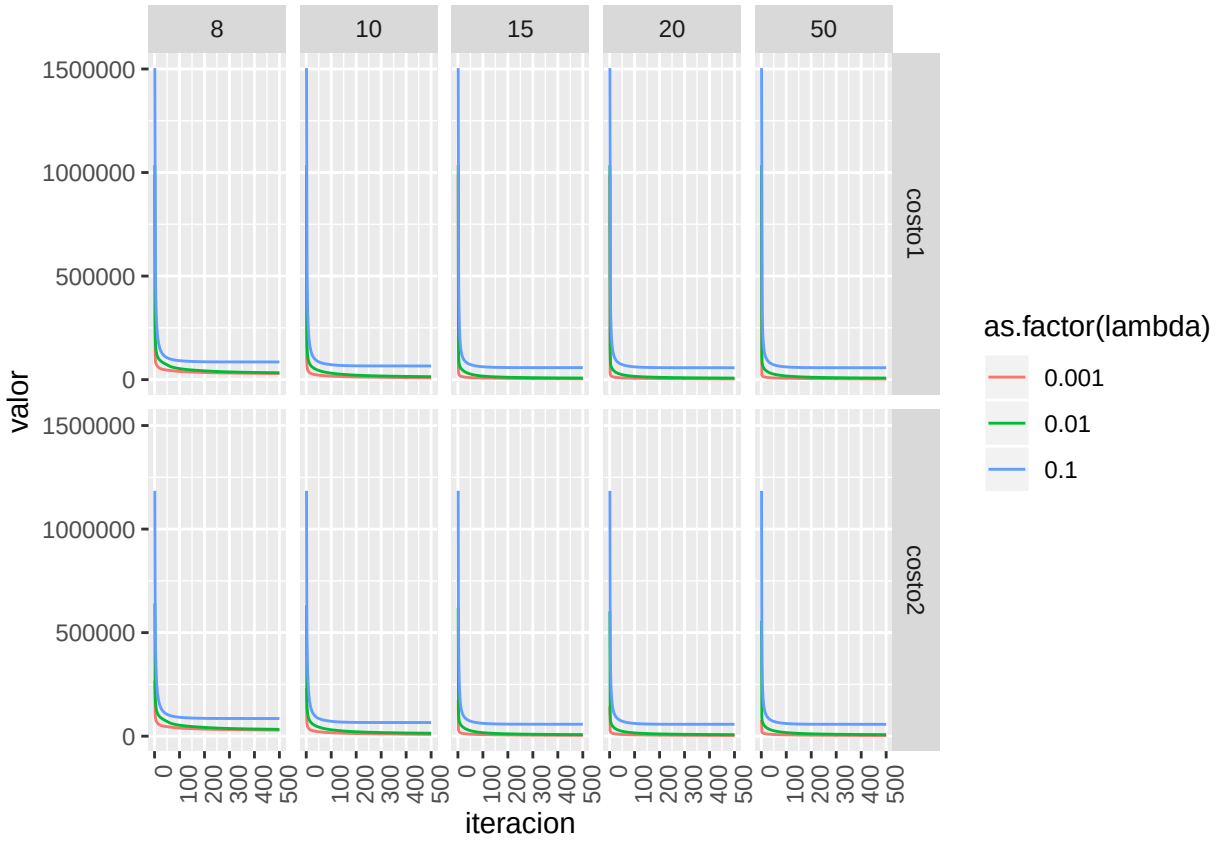


Figure 2: Relación entre la pérdida de la funciones de costos y el número de iteraciones del algoritmo de mínimos cuardados alternados. Los paneles están segementados por número de factores y los colores corresponden al parámetro de regularización

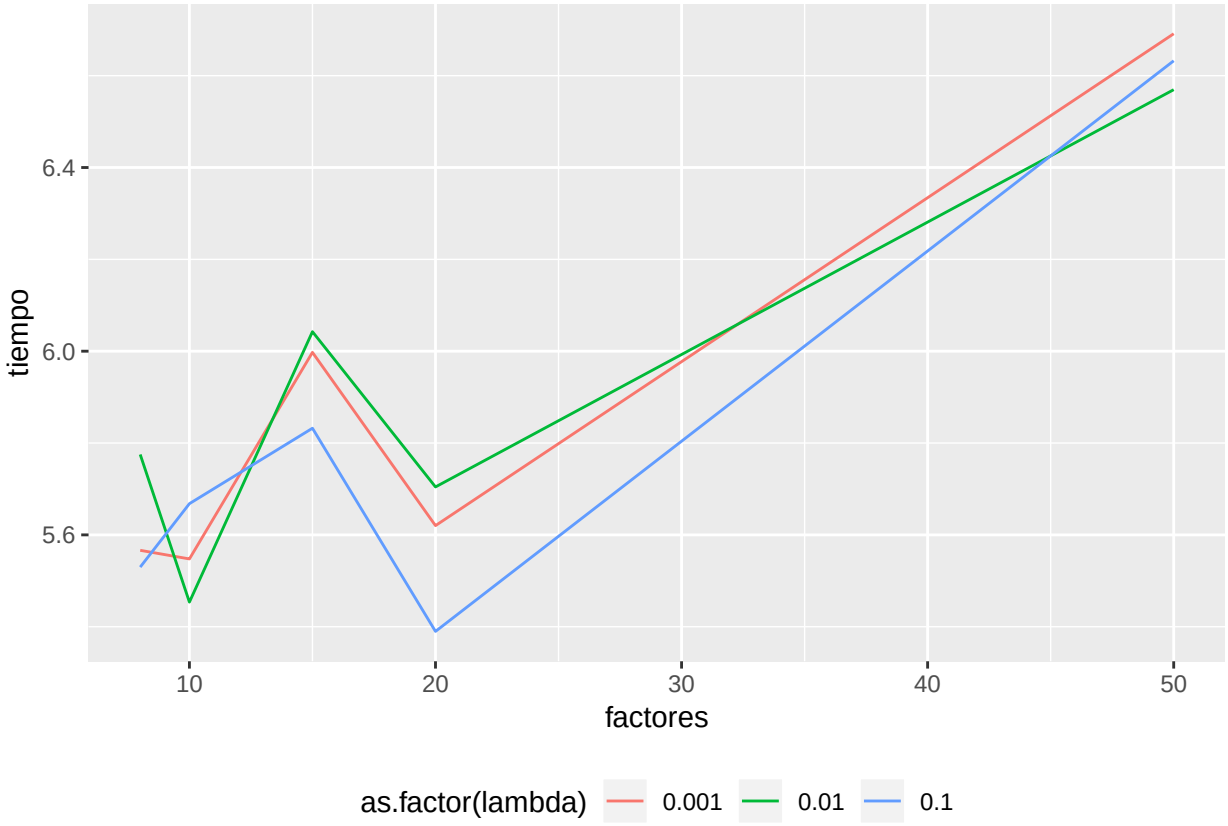


Figure 3: Relación entre tiempo de ejecución del algoritmo y el número de factores. Los colores corresponden al parámetro de regularización

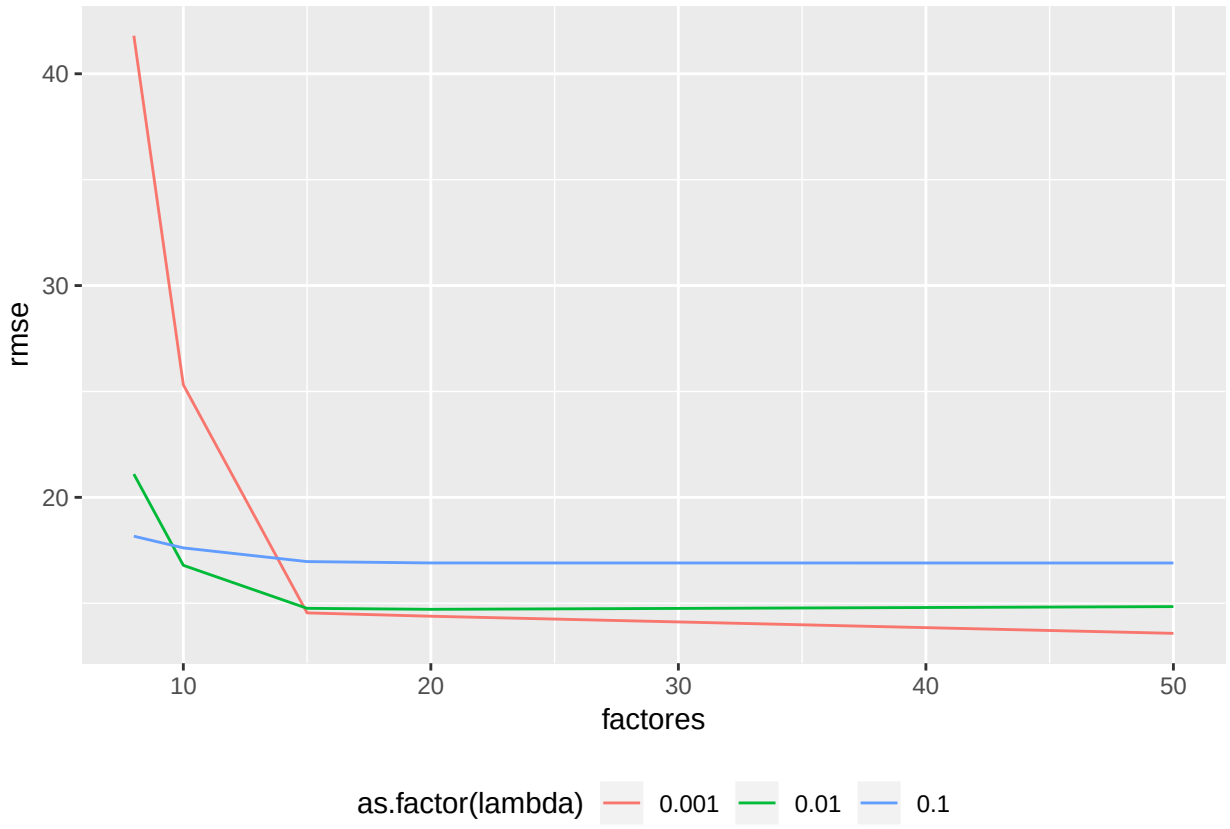


Figure 4: Relación entre el error cuadrático medio y el número de factores y la regularización.

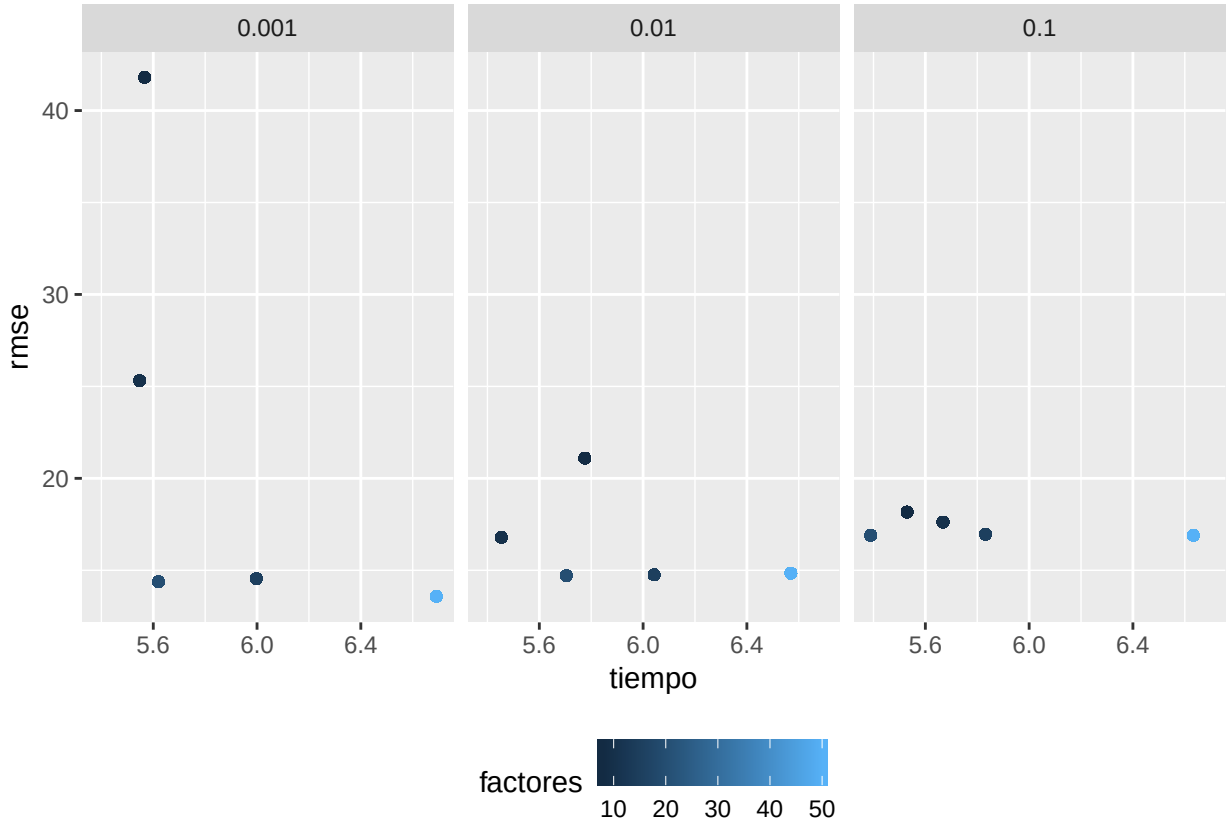


Figure 5: Relación entre el error cuadrático medio y el tiempo de ejerución. Los paneles están segmentados a partir del parámetro de regularización

```
sc <- spark_connect(master = "local")

spark_set_checkpoint_dir(sc, './checkpoint')

# esto normalmente no lo hacemos en R
dat_tbl <- spark_read_csv(sc, name="netflix", "datos/muestra_pelis_todos_usuarios.csv")
dat_tbl <- dat_tbl %>% select(-fecha)
entrena_tbl <- dat_tbl

start_time <- Sys.time()
modelo <- ml_als(entrena_tbl,
              rating_col = 'calif',
              user_col = 'nuevos_usuarios_ids',
              item_col = 'nuevos_ids',
              rank = 50, reg_param = 0.001,
              checkpoint_interval = 5,
              max_iter = 500)

preds <- sdf_predict(entrena_tbl, modelo) %>% collect()
end_time <- Sys.time()
tiempo <- end_time-start_time
print(tiempo)
```

- La implementación del spark para un core con los mismos datos tarda en ejecutarse 2.28 mins.

```
# configuración para spark
config <- spark_config()
#config$`sparklyr.shell.driver-memory` <- "4G"
config$`sparklyr.cores.local` <- 4

# conectar con "cluster" local
sc <- spark_connect(master = "local", config = config)
#sc <- spark_connect(master = "local")

spark_set_checkpoint_dir(sc, './checkpoint')

# esto normalmente no lo hacemos en R
dat_tbl <- spark_read_csv(sc, name="netflix", "datos/muestra_pelis_todos_usuarios.csv")
dat_tbl <- dat_tbl %>% select(-fecha)
entrena_tbl <- dat_tbl

start_time <- Sys.time()
modelo <- ml_als(entrena_tbl,
              rating_col = 'calif',
              user_col = 'nuevos_usuarios_ids',
              item_col = 'nuevos_ids',
              rank = 50, reg_param = 0.001,
              checkpoint_interval = 5,
              max_iter = 500)

preds <- sdf_predict(entrena_tbl, modelo) %>% collect()
end_time <- Sys.time()
tiempo <- end_time-start_time
```

```
print(tiempo)
```

- La implementación para 4 cores tardo marginalmente menos (2.27931 mins), para el mismo conjunto de datos.
- Se puede ver que la implementación de spark tarda casi la mitad de tiempo para ajustar una configuración de 50 variables latentes y una regularización de 0.001

Para un conjunto de datos más grande, como por ejemplo todos los datos de la competencia, la implementación local no funciona ya que se acaba la memoria de la computadora debido a que este algorítmo contiene operaciones matriz a matriz y matriz a vector muy grandes. La respuesta a este problema es la implementación distribuida y paralelizada del algorítmo que se encuentra programada en spark y toma como referencia la misma fuente que la que se utilizó en este proyecto.

# 11 Conclusiones

El algoritmo de mínimos cuadrados alternados regularizado implementado de manera local funciona para matrices de datos pequeñas. La eficiencia de este algoritmo está limitada a la capacidad de procesamiento de una computadora y su memoria. Por estas razones, la principal dificultad que se enfrentó fue la de trabajar con todos los datos de la competencia o con muestras grandes de los datos.

A fin de demostrar el funcionamiento del algoritmo se utilizó una pequeña muestra de los datos de la competencia la cual permitió ejecutar el algoritmo y comparar su desempeño. Los mejores resultados se obtienen en la medida que se incrementa el número de factores, sin embargo, hay un trade off con el tiempo de ejecución. Si se incrementa la regularización se puede reducir el tiempo de ejecución del algoritmo en algunos casos.

Para la muestra de datos el algoritmo ALS implementado de manera local tardó casi el doble que su versión paralelizada y distribuida en Spark para el mismo conjunto de datos. Además, en la medida que la matriz $R$ es más extensa y rala el costo computacional se incrementa considerablemente para la versión local, haciendo poco factible trabajar con datos de gran escala.

Finalmente, es importante mencionar que la calidad de las predicciones mejora en la medida que la matriz $R$ es más grande y se tienen más calificaciones por usuarios y películas.

# 12 Referencias


bugro. (s.f.) **Alternating Least Squares Method for Collaborative Filtering**. Recuperado del WWW el 29 de mayo de 2019 de

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

Bregt, V. (2018). **A latent factor based recommender, solved by the alternating least squares algorithm**. Recuperado del WWW el 29 de mayo de 2019 de https://github.com/mhahsler/recommenderlab/blob/master/R/RECOM_ALS.R

Liao, K. (2018). **Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering**. Recuperado del WWW el 29 de mayo de 2019 de https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square

Zhou, Y. et al (2008). **Large-scale Parallel Collaborative Filtering for the Netflix Prize**. Recuperado del WWW el 29 de mayo de 2019 de 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

Zadeh, R. et al (2015). **Matrix Completion via Alternating Least Square(ALS)**. Recuperado del WWW el 29 de mayo de 2019 de http://stanford.edu/~rezab/classes/cme323/S15/notes/lec14.pdf