La solución de Simon Funk para el premio de Netflix

Sistemas de Recuperación de Información
y de Recomendación

Jacinto Carrasco Castillo.
Anabel Gómez Ríos.

26 de abril de 2017

Profesor: Enrique Herrera Viedma.

Índice

1. Introducción al problema

Los sistemas de recomendación son un tipo de sistema de filtrado de información que proporciona al usuario elementos de un catálogo que pueden resultar de interés. Al contrario que en los sistemas de recuperación de información, donde los resultados dependen de la búsqueda introducida, aquí estamos interesados en mostrar al usuario el producto que mejor se ajusta a sus preferencias, para las cuales nos basaremos en las preferencias mostradas anteriormente por el usuario. Según nos basemos sólo en los elementos del catálogo que le gustan al usuario al que queremos recomendarle otro producto, en otros usuarios con gustos similares para la recomendación del siguiente elemento, o en ambas cosas, estaremos ante sistemas de recomendación basados en contenidos, sistemas de recomendación colaborativos o sistemas híbridos, respectivamente.

Una de las aplicaciones más conocidas es la recomendación de películas, como en la plataforma Netflix.

1.1. Netflix prize

El premio de Netflix fue una competición que buscaba el mejor algoritmo de filtrado colaborativo para predecir valoraciones de películas por usuarios basadas en valoraciones anteriores de otros usuarios sin más información sobre los usuarios ni las películas. En este caso se pedía que el algoritmo fuera colaborativo, de forma que lo que queremos es predecir la valoración que le dará un cierto usuario a una cierta película basándonos en las valoraciones que sí conocemos de otros usuarios. La competición estuvo abierta durante tres años: 2007, 2008 y 2009, dándose el premio final en 2009. El conjunto de entrenamiento que se proporcionó contenía más de cien millones de valoraciones que más de 480.000 usuarios distintos habían dado a 17770 películas. No se daba información acerca de las películas ni de los usuarios, de forma que cada película era un identificador de película y cada usuario era un identificador de usuario. Las valoraciones eran números reales de 1 a 5 y además se proporcionaba la fecha de cada valoración. Si obviamos las fechas de las valoraciones, este conjunto se puede considerar una matriz en la que tenemos las películas en las columnas, los usuarios en las filas y en cada celda las valoraciones. Como es de esperar, no todos los usuarios valoran todas las películas, por lo que esta matriz contiene muchos valores perdidos. De hecho, esta matriz contiene casi nueve mil millones de celdas, y como se ha comentado previamente, sólo se conocían los valores de cien millones, por lo que nos encontramos con una matriz muy dispersa. El conjunto de test contenía casi tres millones de valoraciones, que eran originalmente valores perdidos en la matriz de entrenamiento.

El enfoque que aquí nos ocupa fue el que dio Simon Funk el primer año de la competición, que llegó a ocupar el tercer puesto y fue el único entre los diez primeros que publicó abiertamente qué había hecho en su blog a través de tres entradas [1, 2, 3]. Su solución se basaba en obviar las fechas de las valoraciones y considerarlo una matriz, como hemos comentado aquí, y utilizar la descomposición en valores singulares (SVD por sus siglas en inglés) de la matriz de entrenamiento. Es esta solución la que explicamos de aquí en adelante.

2. SVD

Introducimos en esta sección el método de descomposición de matrices en valores singulares SVD en el que se basó Simon Funk para crear el algoritmo Funk-SVD para la resolución del problema.

La representación elegida es por tanto la de una matriz R donde cada fila se corresponde con un usuario y cada columna con una película, de manera que en la posición $r_{a,i}$ de R tenemos la valoración del usuario a para la película i. Supongamos inicialmente que conocemos todos los valores de la matriz R. Podríamos entonces realizar una descomposición exacta de la matriz usando SVD, $R = U\Sigma V^T$, donde Σ es una matriz diagonal de tamaño $m \times n$, es decir, que sólo tiene valores en la diagonal principal y se rellena con ceros hasta alcanzar la dimensión $m \times n$.

Al igual que se hiciera anteriormente con la recuperación de información, donde cada vez más el interés residía en reconocer conceptos y no sólo términos, en el problema de recomendación se pretenden extraer características latentes comunes entre distintos objetos a recomendar y que podamos asociar a los gustos de los usuarios. Por ejemplo, en el problema que nos ocupa algunas de estas características podrían ser la cantidad de acción en una película o si es o no una película de terror. La descomposición SVD nos permite extraer estas características, aunque es probable que dichas características latentes no se correspondan con conceptos como "acción" o "terror".

Una de las principales características de la descomposición en valores singulares es que las matrices U y V son ortogonales y que la matriz Σ de valores singulares está ordenada de mayor a menor, de forma que en las primeras posiciones de la diagonal principal tenemos los valores singulares más altos, y por tanto, las características más importantes.

Sin embargo, la matrix R contiene más información de la que realmente nos interesa utilizar para realizar predicciones, debido a que guardar tanta información conllevará un tiempo excesivo de cómputo y también a modelos que sobreajustan los datos, con lo que buscaremos una representación más compacta del conocimiento. Esto se consigue mediante una descomposición SVD aproximada, concretamente una aproximación de rango k, donde nos quedamos únicamente con las k características más importantes. Dado que la matriz Σ está ordenada de mayor a menor por los valores singulares, lo único que necesitamos hacer es quedarnos con las k primeras filas y las k primeras columnas, de forma que Σ queda una matriz de tamaño $k \times k$. De esta forma, la formulación de la descomposición es la siguiente:

$$R_k = U_k \Sigma_k V_k^T \ ,$$

donde $R_k \in \mathcal{M}_{m,n}$ es una aproximación de rango k de la matriz original R, $U_k \in \mathcal{M}_{m,k}$, Σ_k matriz diagonal cuadrada de dimensión k y $V_k \in \mathcal{M}_{n,k}$. Esto también nos permitiría reducir el tiempo de cómputo (aunque necesitaríamos establecer el valor apropiado para k).

2.1. Inconvenientes

Valores perdidos Uno de los principales problemas es que para realizar la descomposición, los algoritmos existentes parten de la base de que todos los valores de la matriz son conocidos. Esto podría arreglarse de varias formas, como imputando

previamente los valores perdidos, por ejemplo considerando que los valores perdidos son la media de votaciones por usuario o la media de votaciones por película. Otra forma es normalizar previamente la matriz de forma que tenga media cero, para así considerar que los valores perdidos son cero. Sin embargo en nuestro caso lo que queremos es predecir las valoraciones de algunos de los valores perdidos en esta matriz, que es nuestra matriz de entrenamiento.

Complejidad computacional Otro de los grandes problemas de este algoritmo es el gran tiempo que lleva calcular la descomposición cuando la matriz comienza a tener dimensiones altas. En un problema como este, en el que la matriz tiene casi nueve mil millones de elementos, calcular la descomposición en valores singulares exacta es excesivamente costoso. Además, tendríamos que recalcular la matriz para cada nueva valoración, usuario o película que entrase en el sistema, algo que no nos podríamos permitir. Sin embargo, la matriz no se vuelve a computar en cada ocasión que tenemos una nueva información, sino que las predicciones se calculan mediante un proceso conocido como *fold in*. Esto consiste en que, con la descomposición SVD obtenida previamente, podríamos obtener la relevancia de cada característica para un nuevo usuario o uno que ha realizado una valoración a través de la siguiente fórmula:

$$u_a = V^T r_a \ ,$$

donde r_a es el nuevo vector de valoraciones del usuario a. Entonces, habiendo actualizado U con u_a estamos en disposición de realizar las predicciones. De igual modo, podríamos extraer la influencia de cada característica en un nuevo item dada la valoración de la película por algunos usuarios.

Explicabilidad Estas dimensiones ocultas no siempre se corresponden con conceptos que sean explicables a los usuarios, un aspecto que aporta valor a un sistema de recomendación.

3. Funk-SVD

En esta sección describiremos el algoritmo diseñado por Simon Funk (cuyo nombre real es Brandyn Webb) para la competición de Netflix descrita previamente [3]. Aunque no fue el ganador de la competición en el año en el que participó, 2007, llegó a estar en el top 3 y acabó en el top 10 ese año compitiendo con equipos especializados en sistemas de recomendación.

La influencia de Simon Funk en el reto de Netflix consistió en el uso de SVD como sistema de recomendación y la propuesta de descomposición aproximada más eficiente en tiempo. La idea desarrollada es la de aplicar el gradiente descendiente estocástico para minimizar la función del error obtenido en el conjunto de entrenamiento, con lo que también se solventa el problema de los valores perdidos de la matriz dispersa.

3.1. Gradiente descendente estocástico

El método de gradiente descendente se utiliza cuando se quiere calcular el mínimo de una función determinada. Este algoritmo se basa en partir de un punto determinado x e

ir tomando la dirección del gradiente negativo de la función que se quiere minimizar en pasos proporcionales a dicho gradiente, ya que el opuesto del gradiente señala hacia el mínimo de la función. Para que los pasos sean proporcionales al gradiente se multiplica el mismo por un parámetro λ al que se le llama tasa de aprendizaje o *learning rate*, que es un valor más pequeño que 1. Con esto se pretende evitar que al ser el paso demasiado grande se sobrepase el mínimo.

El método de gradiente descendente estocástico sigue la misma idea pero en lugar de calcular el gradiente en todo el conjunto de entrenamiento de una vez va iterando por los puntos de dicho conjunto y va calculando el gradiente en cada punto para llegar al mínimo. Parte también de un cierto punto x y lo va actualizando en la dirección del gradiente en cada punto. En cada iteración, el paso (el gradiente en un punto determinado) se multiplica también por una tasa de aprendizaje, de forma que la actualización es más pequeña.

Una de las ventajas de este algoritmo es que permite trabajar con valores perdidos, ya que al ir iterando por los puntos del conjunto de train, si un punto contiene un valor perdido es suficiente con no pasar por él en las iteraciones. Esta es también una ventaja del algoritmo de gradiente descendente ya que aunque actualiza los valores con todos los puntos del conjunto de entrenamiento a la vez es también suficiente con coger el subconjunto para el cual se conocen los datos.

3.2. Idea principal del algoritmo Funk-SVD

El principal objetivo de este algoritmo es hacer un cálculo aproximado del SVD de una matriz de dimensiones grandes. Para ello sigue la idea de quedarse con los k primeros valores singulares de la matriz Σ para obtener una aproximación de rango k de la matriz R. Además lo hace de forma iterativa, de forma que se empieza con dos matrices U (de dimensión $m \times k$) y V (de dimensión $n \times k$) inicializadas a un valor pequeño pero no nulo, y se van actualizando en cada iteración para minimizar la raíz cuadrada del error cuadrático medio (RMSE por sus siglas en inglés) mediante una estrategia de gradiente descendente estocástico. El RMSE se calcula con la diferencia entre la matriz R original y UV^T, que es nuestra aproximación. Notemos que al no calcular específicamente la matriz Σ, ésta queda internamente multiplicada en las matrices U y V^T. El valor de k dependerá del número de características latentes que se deseen tener en cuenta, y se iterará tanto en k como en los valores de R. En concreto, para cada característica $j \in \{1, \ldots, k\}$ se itera en los valores de R hasta que se encuentre convergencia en la columna j de U y V y una vez se termina se pasa a la siguiente columna, es decir, a la siguiente característica.

Por tanto, a la hora de calcular U y V podemos simplemente ignorar aquellos valores de R que no son conocidos e iterar en el resto. De igual forma, a la hora de calcular el RMSE sólo tendremos en cuenta aquellos valores que son conocidos en R. Al final, UV^T sí tendrá valores donde R no los tenía. Estas serán nuestras predicciones. Concretamente, para calcular el RMSE en test, se calculará la diferencia entre aquellos nuevos valores de R y los correspondientes de UV^T.

En cuanto al criterio de parada en cada f, la convergencia se puede considerar de varias formas, como iterar hasta que dos errores consecutivos difieran menos que un cierto ε prefijado, iterar hasta que el error, que tiende a cero, esté por debajo de un umbral predijado, o simplemente iterar un número de veces fijado previamente.

En cuanto a las predicciones, un buen punto de partida es considerar las media por película. Sin embargo, esto conlleva el problema de que si una película la han valorado pocos usuarios, algo que es probable puesto que tenemos pocos valores en comparación con la dimensión de la matriz, la media de la película estará muy sesgada a cómo suelan valorar esos pocos usuarios. Es decir, si por ejemplo una película sólo la ha valorado un usuario y le ha puesto un 1, la media de esta película sería un 1. Sin embargo si este usuario suele puntuar por debajo de la media, estaríamos poniendo como media 1 cuando en realidad sabemos que la media será finalmente más alta. Por ello lo que se hace es, para cada predicción $p_{a,i}$ del usuario a de la película i, una media "personalizada" en cada caso, de forma que se tiene en cuenta la media de la película, pero a esta media se le suma un *offset* correspondiente al usuario, de forma que si el usuario a suele valorar por encima de la media, el *offset* sea positivo ya que se espera que este usuario valore por encima de la media esa película. Este *offset* se calcula para cada usuario como la media de las valoraciones de cada usuario menos la media de las medias de las valores de todos los usuarios. Así, este *offset* será positivo si el usuario suele valorar las películas por encima de la media y negativo si las suele valorar por debajo de la media, de forma que al sumárselo a la media de la película, se obtiene una predicción a priori más acertada de la valoración del usuario para una cierta película. De hecho, esta media "personalizada" se utiliza en el algoritmo de forma que las predicciones son $p_{a,i} = \text{media\_pelicula}_i + \text{offset\_usuario}_a + U_{a,\cdot} * V_{i,\cdot}$, y no sólo UV^T.

Como ya hemos comentado, este algoritmo utiliza una estrategia de gradiente descendente estocástico para encontrar el mínimo de la función RMSE. Para ello actualiza las matrices U y V con el gradiente de la función RMSE según U y V en cada caso. Así, la actualización de U y V sería:

$$U_{a,f} = U_{a,f} + \lambda(\varepsilon_{a,i}\ V^T_{f,i}) \ ,$$

$$V^T_{f,i} = V^T_{f,i} + \lambda(\varepsilon_{a,i}\ U_{a,f}) \ ,$$

donde λ es la tasa de aprendizaje, a es el usuario, i la película, f es la característica que se esté aprendiendo en ese momento y $\varepsilon_{a,i}$ es el error de la predicción $p_{a,i}$.

Sin embargo en la actualización de U y V se utiliza además una ténica de regularización para paliar el sobreajuste de las matrices a los datos de entrenamiento, de forma que ésta queda:

$$U_{a,f} = U_{a,f} + \lambda(\varepsilon_{a,i}\ V^T_{f,i} - \gamma U_{a,f}) \ ,$$

$$V^T_{f,i} = V^T_{f,i} + \lambda(\varepsilon_{a,i}\ U_{a,f}) - \gamma V^T_{f,i} \ ,$$

con γ factor de regularización.

3.3. Algoritmo Funk-SVD

Encajando todo lo comentado en las secciones anteriores, ya podemos poner en pseudocódigo el algoritmo final para el cálculo de U y V. Recordemos que la matriz R tiene en las filas a los usuarios y en las columnas a las películas, que tenemos m usuarios y n

películas. Por tanto $R[a, i]$ es la valoración del usuario a sobre la película i.

```
Data: R matriz mxn
Result: U, Vt
U = matriz mxk inicializada a un valor pequeño no nulo
Vt = matriz kxn inicializada a un valor pequeño no nulo
for f en 1,...,k do
    while no haya convergencia do
        for R[a,i] conocido do
            p_ai = media_pelicula[i] + offset_usuario[a] + U[a, ] * Vt[ ,i]
            e_ai = R[a,i] - p_ai
            temp_u = U[a,f]
            U[a,f] = U[a,f] + lambda(e_ai * Vt[f,i] - gamma * U[a,f])
            Vt[f,i] = Vt[f,i] + lambda(e_ai * temp_u - gamma * Vt[f,i])
        end
    end
end
```

Algorithm 1: Funk-SVD

Como se ha comentado previamente, lambda es la tasa de aprendizaje y gamma el factor de regularización.

3.4. Inconvenientes

Elección de los parámetros Este algoritmo deja varios parámetros libres, como el valor o valores a los que se inicializan las matrices U y V, aunque puede ser un solo valor constante y el mismo para U y V. Simon Funk recomienda inicializarlas a 0.1. También se dejan libres los parámetros lambda y gamma, aunque con la restricción, debido a la naturaleza de los mismos, de que también sean números pequeños, alrededor de 0.001 para lambda y alrededor de 0.1 para gamma. Por último, el parámetro k se deja también libre, sin valor recomendado. También se debe elegir un criterio de convergencia o parada, con cuidado para que el algoritmo no tarde mucho en converger. Todos estos parámetros se deberían intentar optimizar para cada problema que se esté intentando resolver, lo que conlleva bastante tiempo de cómputo, ya que aunque este algoritmo es más rápido, las cantidades de datos suelen ser muy grandes.

Pérdida de las propiedades del SVD También hay que tener en cuenta que al ser una aproximación, se pierden propiedades del SVD, como que las matrices U y V sean ortogonales.

4. Código

Hemos implementado en `R` dos versiones distintas del algoritmo Funk-SVD, una con una estrategia de gradiente descendente estocástico y otra con una estrategia de gradiente

descendente. Esto ha sido así porque lo hemos probado con una base de datos de recomendación de películas (más pequeña que el de la competición de Netflix), MovieLens [4], y la variante que sigue una estrategia de gradiente descendente estocástico, al iterar por los datos, tarda más que la que utiliza el gradiente descendente. Además, hemos encontrado, como se comentará en la Sección 5, que en este problema, la estrategia de gradiente descendente da mejores resultados, en cuanto a menor RMSE, que el gradiente descendente estocástico. Hemos decidido utilizar como criterio de parada un número máximo de iteraciones fijado a la hora de llamar al algoritmo.

A continuación se muestra el código, divido en varias funciones: `SGD.feature` calcula una columna de U y V (una característica) con la estrategia de gradiente descendente estocástico, `GD.feature` calcula una característica con la estrategia de gradiente descendente, `baseline` calcula una matriz que en cada posición a, i tiene la media de la película i más el offset del usuario a, es decir, las predicciones base, a las que sumaremos la multiplicación de U por V^T, y `funkSVD`, que calcula las matrices U y V utilizando las funciones anteriores. Esta última acepta como parámetros de entrada la estrategia que se desea utilizar, el valor de la tasa de aprendizaje, del factor de regularización, de k y el número máximo de iteraciones a realizar. Por último, tenemos la función `error.prediction`, que calcula el RMSE en el conjunto de test.

```
#'@function SGD for feature
#'@description Execute stochastic gradient descent on a feature
#' of a ratings matrix and update U and Vt matrices
#'
#'@param R Ratings matrix
#'@param U U matrix in the pseudo SVD decomposition
#'@param Vt t(V) matrix in the pseudo SVD decomposition
#'@param f Feature number to apply convergence
#'@param lr Learning rate
#'@param gamma Regularization term
#'@param baseline Baseline prediction matrix
#'@return List of U and t(V) matrices
SGD.feature <- function(R, U, Vt,f,
                        lr, gamma,
                        baseline){

   not.ai <- which(!is.na(R), arr.ind = T)
   for(j in nrow(not.ai)){
      a <- not.ai[j,1]
      i <- not.ai[j,2]
      r_ai <- R[a,i]
      if(!is.na(r_ai)){
         p_ai <- baseline[a,i] + U[a, ] %*% Vt[ ,i]
         e_ai <- r_ai - p_ai
         temp_u_af <- U[a,f]
         U[a,f] <- U[a,f] +
            lr * (e_ai * Vt[f,i] - gamma * U[a,f])
         Vt[f,i] <- Vt[f,i] +
```

```
            lr * (e_ai * temp_u_af - gamma * Vt[f,i])
      }
   }

   return(list(U = U, Vt = Vt))
}

#'@function GD for feature
#'@description Execute gradient descent on a feature of a ratings
#' matrix and update U and Vt matrices
#'
#'@param R Ratings matrix
#'@param U U matrix in the pseudo SVD decomposition
#'@param Vt t(V) matrix in the pseudo SVD decomposition
#'@param f Feature number to apply convergence
#'@param lr Learning rate
#'@param gamma Regularization term
#'@param baseline Baseline prediction matrix
#'@return List of U and t(V) matrices
GD.feature <- function(R, U, Vt,f,
                       lr, gamma,
                       baseline){

   prediction <- baseline + U %*% Vt
   error <- R - prediction

   temp_u_f <- U[ ,f]
   U[ ,f] <- U[ ,f] + lr *
      (apply(error, 1, function(E_i){
         sum(E_i * Vt[f, ], na.rm = T)) - gamma * U[,f]
      })
   Vt[f,] <- Vt[f,] + lr *
      (apply(error, 2, function(E_a){
         sum(E_a *temp_u_f, na.rm = T)) - gamma * Vt[f,]
      })

   return(list(U = U,Vt = Vt))
}

#'@function Baseline
#'@description Compute Baseline predictions
#'
#'@param R Ratings matrix
#'@return Baseline predictions
baseline <- function(R){
   offset.users <- apply(tra, 1, mean, na.rm = T) -
      mean(apply(tra, 1, mean, na.rm = T), na.rm = T)
   baseline.movies <- apply(tra, 2, mean, na.rm = T)
```

```
   prediction <- matrix(offset.users,
                        ncol = ncol(R), nrow = nrow(R)) +
      matrix(baseline.movies,
             ncol = ncol(R), nrow = nrow(R), byrow = T)
}

#'@function FunkSVD
#'@description Compute approximate SVD decomposition through SGD
#'
#'@param R Matrix of ratings
#'@param k Number of features
#'@param initialization Matrix initialization
#'@param lr Learning Rate
#'@param gamma Regularization term
#'@param epsilon Limit for convergence
#'@return U and V matrices
funkSVD <- function(R, k, initialization = 0.1,
                    lr = 0.001, gamma = 0.02,
                    max.iter = 10,
                    verbose = T,
                    actualization.feature = GD.feature){

   # Initialization of matrix
   U <- matrix(initialization, nrow = nrow(R), ncol = k)
   Vt <- matrix(initialization, nrow = k, ncol = ncol(R))

   # Baseline
   base <- baseline(R)
   # For all the features
   for(f in sample(k)){
      if(verbose)
         print(paste("Training feature", f))
      #Until convergence
      for(l in 1:max.iter){
         sgd.feature <- actualization.feature(R, U, Vt, f,
                                              lr, gamma,
                                              base)
         U <- sgd.feature$U
         Vt <- sgd.feature$Vt
      }
   }

   return(list(U = U,
               V = t(Vt)))
}
```

```
#'@function Error in prediction
#'@description Error in prediction
#'
#'@param original Original matrix
#'@param funk U and V matrices
#'@param train Train matrix to regularize
#'@return RMSE
error.prediction <- function(original, funk, train){
   # Baseline preduction
   baseline <- baseline(train)
   prediction <- baseline + funk$U %*% t(funk$V)
   # RMSE
   sqrt(mean((original-prediction)^2, na.rm=T))
}
```

5. Experimentos

Los experimentos se han llevado a cabo con el conjunto pequeño de datos de películas y valoraciones de MovieLens. Este conjunto tiene 100004 predicciones de 671 usuarios sobre 9066 películas [4]. Este conjunto de datos también incluye los *tags* asociados a una película, aunque se ha optado por obviar esta información ya que las características ocultas pretenden realizar de manera automática esta síntesis de las películas. Puesto que sólo se incluye una valoración por la tupla *(película, usuario)*, se ha optado por descartar la información del momento en el que se ha realizado la valoración.

Para la realización de los experimentos se ha realizado una partición en train y test al $70-30\,\%$ de las valoraciones, dotando tanto al conjunto de entrenamiento como al de test la forma de matriz dispersa descrita previamente.

Los algoritmos programados se han ejecutado con los siguientes parámetros:

Tabla 1: Parámetros utilizados

Parámetro	Descripción	Valores
K	Número de características ocultas	2,6,10,...,38
λ	Tasa de aprendizaje	0.0005, 0.001, 0.0015, 0.002
γ	Regularización	0.01, 0.02, 0.03
`max.iter`	Número de iteraciones	10, 50

Mostramos los resultados obtenidos en la siguiente gráfica, la cual nos aporta información relevante sobre la eficiencia de los dos algoritmos probados y de los parámetros. En la Figura 1 se observa que el algoritmo SGD tiene siempre un RSME mayor que el el gradiente descendiente calculado de forma matricial. También se observa que el error disminuye conforme aumenta la tasa de aprendizaje, diferencia que se acrecenta a mayor número de iteraciones. Con respecto al número de características, observamos que el error de entrenamiento permanece estable a partir de $k = 20$, con lo que dejaríamos de probar con k superiores, pues estaríamos complicando en exceso el modelo.

Figura 1: Error en train

En la Figura 2 se observa cómo el valor de regularización no afecta al error obtenido para ninguna combinación de algoritmo, número de iteraciones y tasa de aprendizaje.

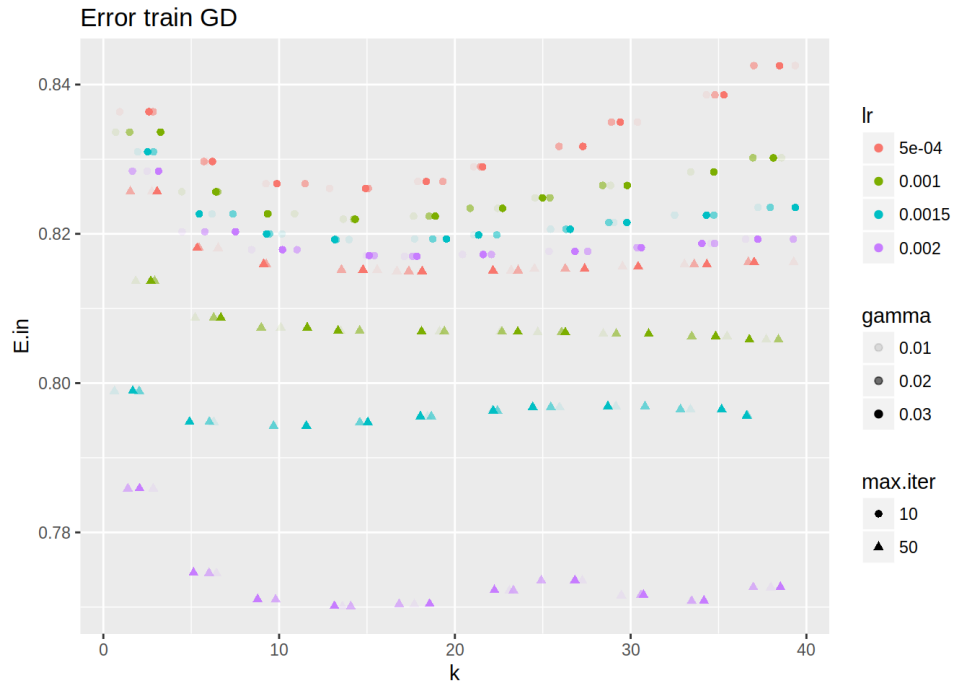

Figura 2: Error según la tasa de regularización

Por último, en la Figura 3 mostramos la comparación del error de entrenamiento frente

al de test para una combinación concreta de parámetros. Aquí podemos observar que el error de entrenamiento es superior al de test, situándose en torno a 0.93, y que, aunque el error de entrenamiento se veía muy afectado por las iteraciones realizadas, en test son prácticamente idénticos para k menor que 20, cuando empieza a subir también el error de test para 10 iteraciones. Aunque este conjunto de datos es mucho más pequeño que el usado en el Premio de Netflix, obtenemos un error ligeramente superior que el obtenido por Simon Funk en la competición (0.8921), sin embargo, el error representa una mejora que sí que es relevante con respecto a la predicción básica basada en la media de cada película y el *offset* de cada usuario con la media de los usuarios, que tiene un RMSE de 1,5 en test (y un poco menos en train, en torno a 1,45 debido a la centralización de los datos).

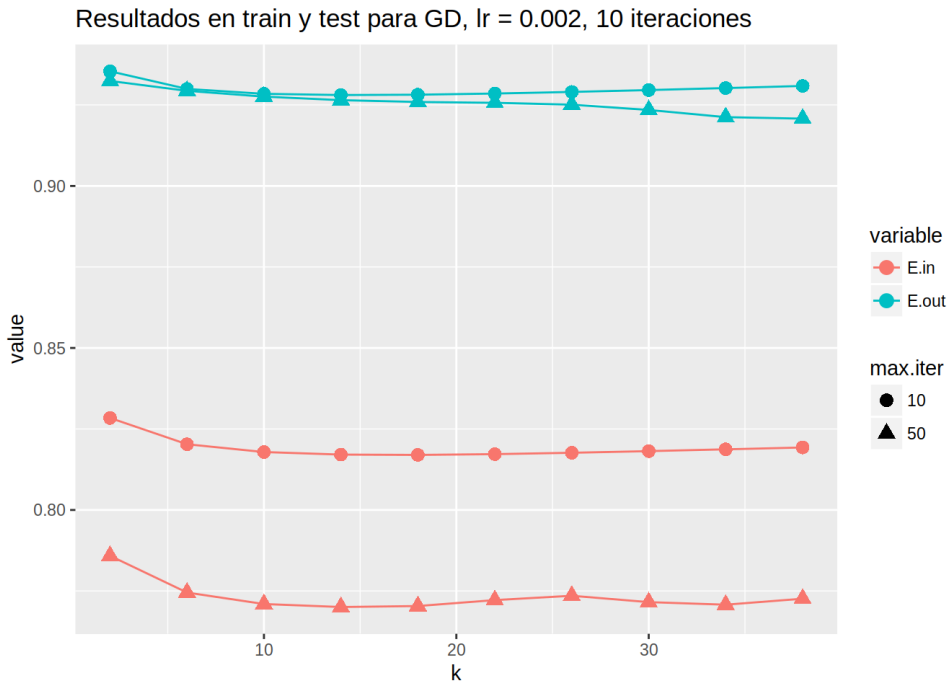

Figura 3: Error de entrenamiento frente a test

Referencias

[1] Simon Funk Journal: Netflix Challenge. `http://sifter.org/~simon/journal/20061027.2.html`. Último acceso: 26/04/2017.

[2] Simon Funk Journal: Netflix Update: Don't Try This at Home. `http://sifter.org/~simon/journal/20061102.1.html`. Último acceso: 26/04/2017.

[3] Simon Funk Journal: Netflix Update: Try This at Home. `http://sifter.org/~simon/journal/20061211.html`. Último acceso: 26/04/2017.

[4] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.