Ferretti Valerio 03/01/2017

MÁSTER UNIVERSITARIO EN INGENIERÍA Y TECNOLOGIA DEL SOFTWARE.

Memoria de trabajo Minería de Datos Aplicada

1) Introducción y Reto seleccionado

El objetivo del presente trabajo es lo de proporcionar predicciones sobre la evaluación y el nivel de agradecimiento de usuarios sobre películas que todavía no han visto basándose en sus evaluaciones de películas vistas previamente. Los ficheros que constituyen nuestra base de datos han sido bajado desde Kaggle a la pagina https://inclass.kaggle.com/c/predict-movie-ratings.
Específicamente Kaggle proporciona 3 archivos:

a) El fichero **TRAIN_V2** contiene 750156 líneas cada una hecha por 4 atributos: ID, USER_ID, MOVIE_ID, RATING. Los primeros tres son números que identifican unívocamente una línea del archivo TRAIN_V2 (evaluación de un usuario sobre la película), un usuario y una película. El último atributo es una evaluación de la película donde el valor mínimo es 1 y el máximo es 5. El objetivo de este fichero sería entrenar nuestro modelo de predicción de manera que sea posible prever las evaluaciones de los mismo usuarios sobre nuevas películas.
b) El fichero **TEST_V2** contine 250053 líneas cada una hecha por tres atributos: ID, USER_ID, MOVIE_ID. Para cada línea hace falta prever el nivel de agradecimiento del usuario USER_ID sobre la película MOVIE_ID basándose en sus previas evaluaciones en el fichero TRAIN_V2.
c) El fichero **SampleSubmission** es un modelo de cómo debería ser el archivo que tenemos que subir a Kaggle. Este fichero objetivo tiene dos campos: un campo ID y una evaluación entre 1 y 5. El campo ID se refiere a una línea en el archivo TEST_V2. El campo evaluación sería el

nivel de agradecimiento previsto para el usuario USER_ID sobre la película MOVIE_ID en el fichero TEST_V2.

2) Estado del arte de las técnicas utilizadas

3.1) Recommender System

Los **Recommender System** (o Recommendation System) son sistemas para filtrar las informaciones cuyo objetivo es prever el rating o la preferencia que un usuario va a exprimir sobre un objeto. En la mayoría de las veces estos sistemas son empleados en la predicciones de agradecimiento de peliculas, musica, libros y productos en general.
Típicamente los recommender system producen una lista de recomendaciones para cada usuario en dos maneras distintas: a través del Content-Based Filtering o a traves del Collaborative Filtering.

3.2) Content-Based Filtering VS Collaborative Filtering

El Content-Based Filtering analiza las características de los objetos que han sido evaluado por el usuario para producir una lista de objetos que podrian gustar al mismo usuario porque comparten dichas características con los objetos que el usuarios ya ha evaluado positivamente.
El Collaborative Filtering construye un modelo basándose en el comportamiento pasado del usuario (objetos previamente comprados y previamente seleccionados/evaluados) y basándose en la historia de otras personas cuyo comportamiento se puede considerar similar a dicho usuario. Típicamente lo que hace un algoritmo de Collaborative Filtering es:

a) Un usuario expresa su evaluación sobre un ítem; esa evaluación puede ser considerada como una representación aproximada del interés de ese usuario en un dominio específico.
b) El sistema confronta las preferencias de los usuarios y encuentra usuarios con gustos parecidos.
c) Cuando el sistema tiene un a lista de usuarios similares aconseja a un usuario un ítem que ha sido evaluado positivamente por un usuario similar y que todavía no han sido seleccionado por el usuario en cuestión.

Es claro entonces cuál es la verdadera diferencia entre las dos técnicas: la primera se focaliza en las características de los productos seleccionados para aconsejar producto similares, la segunda aconseja objetos basándose en la historia y en el comportamiento del usuario y de otros usuarios similares a el.

3.3) Ventajas y Desventajas

Obviamente las dos técnicas tienen ventajas y desventajas.
El problema del Collaborative-Filtering es el comienzo de la elaboración; este problema es conocido como el cold start problem. Necesita muchas informaciones sobre los usuarios para empezar a proporcionar previsiones acuradas y la ausencia de dichas informaciones perjudica su correcta ejecucion. También muchas veces tenemos millones de usuarios y productos y los algoritmos de Collaborative Filtering necesitan muchas potencia de cálculo y memoria. La análisis y la comparación del comportamiento de los usuarios en este proyecto con una base de datos relativamente pequeña (6040 usuarios) ha tardado 6 o 7 minutos!
Al revés, el Content-Based Filtering no necesita muchas informaciones sobre los usuarios sino sobre los ítems. En general no necesita muchas informaciones para empezar a proporcionar recomendaciones pero sus posibilidades son limitadas a similaridades entre ítems y siempre consejera items similares a los primeros evaluados por el usuario.

4) Implementación de los algoritmos de Collaborative Filtering en R

En este proyecto vamos a utilizar la técnica del Collaborative Filtering para estimar las evaluaciones de usuarios sobre nuevas películas. El paquete de R 'recommenderlab' nos proporciona una implementación de los algoritmos de collaborative filtering. A través de ese paquete vamos a utilizar dos algoritmos diferentes: **Item-Based Collaborative Filtering** (IBCF) y **User-Based Collaborative Filtering** (UBCF).

a) Item-Based Collaborative Filtering

- Organizamos los datos en una hoja de datos donde cada línea corresponde a un usuario, cada columna a un producto.
- Calcular las similaridades entre cada pareja de productos, es decir calcular las similaridades entre cada columna de la hoja de datos. Para

conseguir eso empleamos la “cosine similarity”: para cada diferente pareja de columna en la hoja de datos ejecutamos las siguiente operación

$$cosine.similarity \ <- \ sum(x * y) \ / \ (sqrt(sum(x * x)) \ * \ sqrt(sum(y * y)))$$

donde ‘x’ y ‘y’ son dos vectores y cada ocurrencia de sum(x*y) es un producto escalar entre los dos. El valor obtenido es una medida de similaridad entre los dos producto obtenida considerando las evaluaciones de todos los usuarios sobre dicho producto. 1 significa igualdad perfecta, 0 ortogonalidad (ninguna correlación).

- Guardamos los resultados del paso precedente en una hoja de datos donde filas y columnas corresponden a productos.
- A usuarios que ha seleccionado un producto X podemos entonces aconsejar el producto en la fila correspondiente más similar al producto X ya evaluado positivamente. Ese producto sería el cuyo valor en la matriz es más cercano a 1.

a) User-Based Collaborative Filtering

- Organizamos los datos en una hoja de datos donde cada línea corresponde a un usuario, cada columna a un producto.

Para cada usuario:

- Calculamos la similaridad con todos los otros usuarios a través de la cosine similarity. Cada usuario es una fila de la matriz o hoja de datos con tantos campos cuanto son los productos. Obtenemos entonces un vector de valores que miden la similaridad del usuario en cuestión con todos los otros.
- Creamos una matriz/hoja de datos que contiene las evaluaciones de todos los usuarios sobre todos los producto que todavía no han sido seleccionados por el usuario en cuestión.
- Multiplicamos el vector de similaridades con la matriz recién obtenida y conseguimos crear una matriz de evaluaciones de productos todavía no evaluado por el usuario en cuestión pasada a través del vector de similaridades.

- Para cada columna de la matriz recién obtenida sumamos los valores de todas sus celdas y obtenemos entonces un valor para cada producto todavía no evaluado por el usuario en cuestión.
- Para cada producto (no evaluado por el usuario en cuestión) sumamos todos los valores en el vector de las similaridades que corresponden a usuarios que han evaluado este especifico producto. Dividimos entonces los valores obtenidos en el paso precedente con los valores obtenido a través de la suma de las celda del vector de similaridades. El valor obtenido para cada producto es la evaluación estimada que el usuario en cuestión pondría al seleccionar el producto.

Se queda entonces claro porque el primer método es llamado Item-Based y el segundo User-Based.
El primero calcula un matriz de similaridad entre objetos a través de productos escalares entre vectores que son la historia de evaluacion de ese producto hecha por todos los usuarios.
El segundo método calcula las similaridades entre cada pareja de usuarios y utiliza esa matriz para pesar las evaluaciones que otros usuarios han dado sobre un producto. Las evaluaciones de usuarios similares al usuario en cuestión tomarán un papel más importante en la estima final de agradecimiento.

3) Preprocesamiento y generación de modelos

Voy ahora a presentar el código R y a comentar las instrucciones con las cuales he aplicado ambas las técnicas para prever las evaluaciones de los usuarios.

El paquete 'recommenderlab' es el paquete más importante porque contiene las implementaciones de las funciones de Collaborative Filtering. El paquete 'reshape2' va ser necesario en fase de preprocesamiento para poner los datos en la formas correcta y 'ggplot2' hace falta para visualizar los datos en forma gráfica.

```
####Reference: https://inclass.kaggle.com/c/predict-movie-ratings

# Estableciendo la crpeta de trabajo
```

```
setwd("E:/Universita/Mineria de Datos Aplicada/Trabajo/NUOVO_TRABAJO")

# Instalando los paquetes y cargando las librerias
install.packages('recommenderlab')
install.packages('reshape2')
install.packages('ggplot2')
library(recommenderlab)
library(reshape2)
library(ggplot2)

# Guardando el archivo de entrenamiento en la hoja de datos tr
tr<-read.csv("train_v2.csv",header=TRUE)
# Vistazo de las primeras líneas en la hoja de datos
head(tr)
```

	ID	user	movie	rating
1	610739	3704	3784	3
2	324753	1924	802	3
3	808218	4837	1387	4
4	133808	867	1196	4
5	431858	2631	3072	5
6	895320	5410	2049	4

PRE PROCESAMIENTO

- Eliminar desde la hoja de datos tr, donde hemos importando el archivo train_v2.csv, la primera columna ID que no es necesaria para el procesamiento.

```
# Eliminar la columna ID
tr<-tr[,-c(1)]
# Controlar si ha sido eliminado correctamente
tr[tr$user==1,]
```

- La función acast() pertenece al paquete 'reshape2' y hace falta para convertir la hoja de datos 'tr' en una forma que sea útil a la elaboración. El output del comando es otra hoja de datos donde cada línea corresponde a un usuario y cada columna a un película. El contenido de la hoja de datos es la evaluación que que un usuario específico a dado a una película específica. Obviamente la mayoría de las celda tendrá como valor NA (not available) porque la mayoría de los usuarios han evaluado sólo un subconjunto pequeño de las películas. El primer parámetro de la función es la hoja de datos, el segundo una fórmula que especifique qué atributo poner como fila y que atributo poner como columna.

```
# Conversion a traves de acast():
#       m1  m2   m3   m4
# u1    3   4    2    5
# u2    1   6    5
# u3    4   4    2    5
g<-acast(tr, user ~ movie)
g[1,1:100]

# Conversión en una matriz
R<-as.matrix(g)
```

- Convertimos entonces la matriz obtenida en una 'RealRatingMatrix'. Esa es una estructura definida en el paquete 'recommenderlab' y necesaria para poder generar el modelo a través de la función Recommender() que la toma como parámetro.

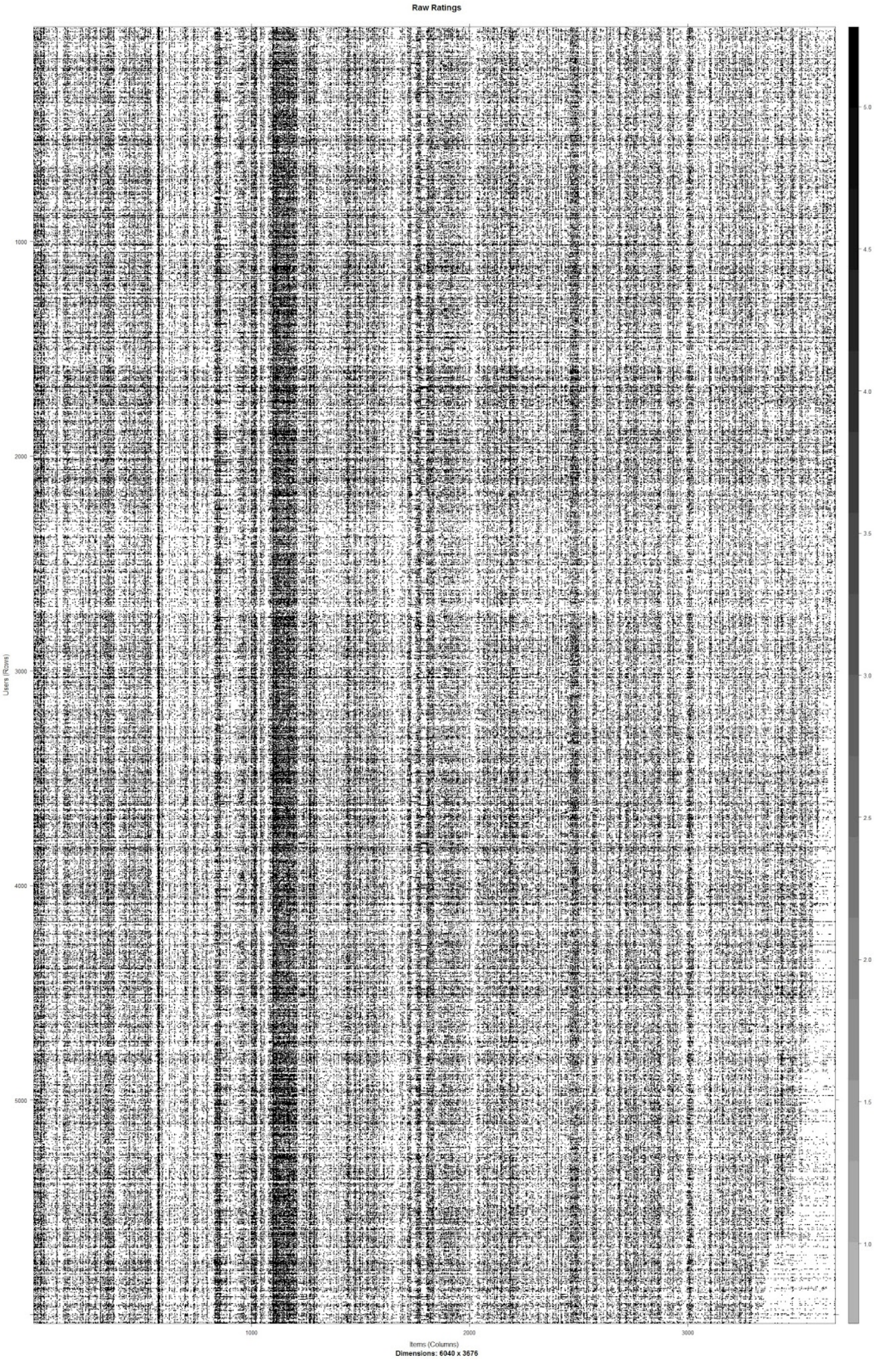
```
# Conversion a una realRatingMatrix
r <- as(R, "realRatingMatrix")
r

# Visualización de r como lista, matriz y hoja de datos
as(r, "list")     # A list
as(r, "matrix")   # A sparse matrix
head(as(r, "data.frame"))
```

GENERACIÓN DEL MODELO

- Presentación de los datos en forma gráfica a través de la función image(). En la imagen cada columna representa una pelicula y cada riga representa un usuario. Columnas negras corresponden a películas vistas por muchos usuarios y evaluadas positivamente, filas negras corresponden a usuarios que han evaluado muchas películas positivamente. Las evaluaciones se traducen en diferentes escalas de gris como indicado a la derecha de la imagen.

```
image(r, main = "Raw Ratings")
```


Raw Ratings
Users (Rows)
1000
2000
3000
4000
5000
1000
2000
3000
Items (Columns)
Dimensions: 6040 x 3676
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0

- El paso siguiente es la creación del modelo. Los comandos abajo crean un recommender object a través de la instrucción Recommender() especificado el método que queremos emplear (UBCF o IBCF) y la topología de similaridad que queremos (cosine similarity). El comando predict() rellena una matriz que contiene la previsiones de evaluaciones para cada pareja user - item. El parámetro 'type' es usado para especificar si queremos las evaluaciones previstas o si queremos obtener simplemente el número n de objetos cuya evaluación prevista es más alta (predict(rec, r[1:nrow(r)], type="topNList", n=10)).

```
#Metodo User-Based Collaborative Filtering
rec=Recommender(r[1:nrow(r)],method="UBCF",param=list(normalize="Z-score",method="Cosine"))

#Metodo Item-Based Collaborative Filtering
rec=Recommender(r[1:nrow(r)],method="IBCF",param=list(normalize="Z-score",method="Cosine"))

recom <- predict(rec, r[1:nrow(r)], type="ratings")
recom
```

- Lo que hace falta hacer es leer al archivo testv2_csv y crear una hoja de datos que contenga las evaluaciones previstas para todas las pareja userID - movieID en dicho archivo. El código siguiente abre ese archivo y lo guarda en una hoja de datos. El bucle for la escanea y crea otra hoja de datos 'ratings' rellenandola con las evaluaciones que subir a kaggle.

```
test<-read.csv("test_v2.csv",header=TRUE)
head(test)

# Convertir la hoja de datos recom en una lista
rec_list<-as(recom,"list")
head(summary(rec_list))

# Creación de la hoja de datos 'rating' que va a contener las evaluaciones que subir en
# Kaggle
ratings<-NULL


# Bucle que escanea toda las filas de la hoja de datos test
for ( u in 1:length(test[,2]))
{
   # Leer userID y movieID desde test
   userid <- test[u,2]
   movieid<-test[u,3]
```

```
  # Obtener lista recomendaciones para userid y transformala en hoja con una columna
  u1<-as.data.frame(rec_list[[userid]])

  # Añadimos una segunda columna a u1 con los ID de las películas de manera que haya
  # correspondencia entre evaluación (columna 1) y movieID (columna 2).
  # Conseguimos eso a través de la función row.names(u1).
  u1$id<-row.names(u1)

  # Accedemos a la evaluación prevista para la película movieid para el usuario userid
  x= u1[u1$id==movieid, 1]

  # Si la película para la cual estamos buscando una evaluación prevista para el usuario
  # userid no ha sido evaluada por nadie y no estaba entonces en el archivo train_v2.csv,
  # asignamos un valor mediano que seria 2. Si encontramos una evaluación la añadimos a la
  # hoja de datos ratings en un nueva fila de la primera columna.
  if (length(x)==0)
  {
    ratings[u] <- 2
  }
  else
  {
    ratings[u] <-x
  }

}
```

- El último paso es la creación del archivo excel submit.csv que subir en kaggle con los campos ID, evaluación.

```
length(ratings)

# Creación hoja de datos ID, ratings
tx<-cbind(test[,1],round(ratings))

# Escritura archivo csv en la working directory
write.table(tx,file="submitfile.csv",row.names=FALSE,col.names=FALSE,sep=',')
```

4) Evaluación y consideraciones finales

Vamos ahora a presentar el código empleado para evaluar y confrontar los modelos creados: UBCF y IBCF. Otra vez disfrutamos de las funciones proporcionadas por el paquete 'recommenderlab' para encontrar el Root Mean Square Error, Mean Square Error y Mean Absolute Error. Para conseguirlo

vamos a dividir la hoja de datos ‘r’ en dos trozos, uno para el entrenamiento del modelo y una para la evaluación del mismo modelo.

```
# creacion de un esquema de evaluacion donde el 90% de los datos son empladoes para el entrenamiento del model y el 10% para la evaluacion del modelo
e <- evaluationScheme(r[1:nrow(r)], method="split", train=0.9, given=9)

Rec.ubcf <- Recommender(getData(e, "train"), method="UBCF", param=list(normalize = "Z-score",method="Cosine"))

Rec.ibcf <- Recommender(getData(e, "train"), method="IBCF", param=list(normalize = "Z-score",method="Cosine"))

p.ubcf <- predict(Rec.ubcf, getData(e, "known"), type="ratings")

p.ibcf <- predict(Rec.ibcf, getData(e, "known"), type="ratings")

# Creacion de la hoja de datos error y calculo de las medidas de error
error.ubcf<-calcPredictionAccuracy(p.ubcf, getData(e, "unknown"))
error.ibcf<-calcPredictionAccuracy(p.ibcf, getData(e, "unknown"))
error <- rbind(error.ubcf,error.ibcf)
rownames(error) <- c("UBCF","IBCF")
error
```

```
          RMSE      MSE       MAE
UBCF 1.204530 1.450894 0.9566034
IBCF 1.128218 1.272876 0.7978596
```

Desde esa comparación de los modelos parece que lo mejor sea el modelo IBCF porque presenta valores menores en todos los campos. De todas formas los errores parece un poco mas alto de lo que se podía esperar con un Mean Absolute Error de 0.79 en el caso de IBCF.

5) Codigo completo

```
####Referencia: https://inclass.kaggle.com/c/predict-movie-ratings

setwd("E:/Universita/Mineria de Datos Aplicada/Trabajo/NUOVO_TRABAJO")

install.packages('recommenderlab')
install.packages('reshape2')
install.packages('ggplot2')
library(recommenderlab)
library(reshape2)
library(ggplot2)

tr<-read.csv("train_v2.csv",header=TRUE)
head(tr)
```

```
############ Preprocesamiento ############

tr<-tr[,-c(1)]
tr[tr$user==1,]

g<-acast(tr, user ~ movie)
g[1,1:100]
R<-as.matrix(g)
r <- as(R, "realRatingMatrix")
r

as(r, "list")
as(r, "matrix")
head(as(r, "data.frame"))

image(r, main = "Raw Ratings")

############## ANALISIS y CREACION DE MODELO ############
rec=Recommender(r[1:nrow(r)],method="UBCF",param=list(normalize="Z-score",method="Cosine"))
rec=Recommender(r[1:nrow(r)],method="IBCF",param=list(normalize="Z-score",method="Cosine"))

recom <- predict(rec, r[1:nrow(r)], type="ratings")
recom

test<-read.csv("test_v2.csv",header=TRUE)
head(test)
rec_list<-as(recom,"list")
head(summary(rec_list))
ratings<-NULL

for ( u in 1:length(test[,2]))
{
   userid <- test[u,2]
   movieid<-test[u,3]

   u1<-as.data.frame(rec_list[[userid]])
   u1$id<-row.names(u1)
   x= u1[u1$id==movieid,1]

   if (length(x)==0)
   {
     ratings[u] <- 2
   }
   else
   {
     ratings[u] <-x
   }

}
length(ratings)
tx<-cbind(test[,1],round(ratings))

write.table(tx,file="submitfile_IBCF.csv",row.names=FALSE,col.names=FALSE,sep=',')
```

```
########## Model Evaluation ######################
e <- evaluationScheme(r[1:nrow(r)], method="split", train=0.9, given=9)

Rec.ubcf  <-  Recommender(getData(e,  "train"),  method="UBCF",  param=list(normalize  =
"Z-score",method="Cosine"))
Rec.ibcf  <-  Recommender(getData(e,  "train"),  method="IBCF",  param=list(normalize  =
"Z-score",method="Cosine"))
p.ubcf <- predict(Rec.ubcf, getData(e, "known"), type="ratings")
p.ibcf <- predict(Rec.ibcf, getData(e, "known"), type="ratings")

error.ubcf<-calcPredictionAccuracy(p.ubcf, getData(e, "unknown"))
error.ibcf<-calcPredictionAccuracy(p.ibcf, getData(e, "unknown"))
error <- rbind(error.ubcf,error.ibcf)
rownames(error) <- c("UBCF","IBCF")
error
```