Movie Recommender Systems

Jason Michaels

12/29/2016

R Markdown

Load the required packages:

```r
library(irlba)
```

```
## Loading required package: Matrix
```

```r
library(readr)
library(reshape2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(data.table)
```

```
## -------------------------------------------------------------------------

## data.table + dplyr code now lives in dtplyr.
## Please library(dtplyr)!

## -------------------------------------------------------------------------

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, last

## The following objects are masked from 'package:reshape2':
##
##     dcast, melt
```

```r
library(Matrix)
library(skmeans)
library(qlcMatrix)
```

```
## Loading required package: slam

## Warning: package 'slam' was built under R version 3.3.2
```

```r
library(fields)
```

```
## Loading required package: spam
```

```
## Loading required package: grid

## Spam version 1.4-0 (2016-08-29) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: maps
```

```r
library(ggplot2)
```

```r
set.seed(1)

# Import data and make a few quick edits

ratings <- read_delim("/Users/Jason/Desktop/Machine Learning/Movie-Lens/Data/ratings.dat",
                      col_names = FALSE, delim = ":")
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   X2 = col_character(),
##   X3 = col_integer(),
##   X4 = col_character(),
##   X5 = col_integer(),
##   X6 = col_character(),
##   X7 = col_integer()
## )
```

```r
head(ratings)
```

```
## # A tibble: 6 × 7
##      X1    X2    X3    X4    X5    X6        X7
##   <int> <chr> <int> <chr> <int> <chr>     <int>
## 1     1  <NA>  1193  <NA>     5  <NA> 978300760
## 2     1  <NA>   661  <NA>     3  <NA> 978302109
## 3     1  <NA>   914  <NA>     3  <NA> 978301968
## 4     1  <NA>  3408  <NA>     4  <NA> 978300275
## 5     1  <NA>  2355  <NA>     5  <NA> 978824291
## 6     1  <NA>  1197  <NA>     3  <NA> 978302268
```

```r
ratings <- ratings[,c(1,3,5)]
names(ratings) <- c("userId", "movieId", "rating")
```

Create a sparse matrix with the i jth entry indicating the rating of user i of movie j

```r
data = sparseMatrix(as.integer(ratings$userId), as.integer(ratings$movieId), x = ratings$rating)
dim(data)
```

```
## [1] 6040 3952
```

Note that our matrix has more columns than there are individual movies in the ratings data. This is because some movies are missing ratings. If movie ID i is not rated, then the corresponding row in the data set is missing. This is fine and doesn't affect our analysis.

Sample 1/10th of the nonzero entries in our data. Put the indices in a list called test_ind. The ith element of the list contains the indices for the ratings we will take out of the training set for user i

```
non_zero <- apply(data, 1, function(x) which(x != 0))
test_ind <- lapply(non_zero, function(x) sample(x, floor(length(x)/10)))

# We will use the non-zero elements listed in test_ind to filter out the test items from
# the original ratings dataset.
ratings_train <- as.data.table(ratings)
setkey(ratings_train, movieId)

for(row in 1:length(test_ind)){
  # if(row %% 10 == 0){print(row)}
  ratings_train <- ratings_train[!(userId == row & movieId %in% test_ind[[row]])]
}

# Since some users tend to rate movies they like higher than others, we will scale each
# rating by the mean rating of the user.
means <- ratings_train %>% group_by(userId) %>% summarize(mean(rating))
ratings_train <- merge(ratings_train, means, by = "userId")
names(ratings_train)[4] <- "means"

ratings_train$new_rating = ratings_train$rating - ratings_train$means

train_scaled = sparseMatrix(as.integer(ratings_train$userId), as.integer(ratings_train$movieId), x = ra
```

We now have a sparse matrix (train_scaled), where the i jth entry represents user i's scaled rating for film j.

Now we will get a similarity matrix between users. For our similarity metric, we will use cosine distance rather than Euclidean. This allows us to ignore absent ratings.

```
train_T <- t(train_scaled)
similarity <- cosSparse(train_T)

# We can use data tables to try and speed things up a bit
sim_frame <- as.data.table(as.matrix(similarity))

# Come up with a matrix of ORDERED similarities for each user
ordered_sim <- apply(similarity, 2, function(x) order(x, decreasing = TRUE))
ordered_sim <- as.data.table(ordered_sim)
```

Now we have all the information we need to run KNN. Let's do this for several values of k.

```
ptm <- proc.time()

k <- 1:40
SSE <- rep(0, length(k))
for(user in 1:nrow(data)){ # Go through each user individually
  if(user %% 1000 == 0){print(paste0("USER ", as.character(user), ": ", as.character((proc.time() - ptm
  userId <- ordered_sim[[user]] # Rank of most similar user to current one
  rank = 1:length(userId)  #
  # Turn this into a dataset with the 1:n_user indicating their rank, and then userid
  sim_rank = data.table(userId, rank)
  preds <- matrix(rep(rep(0, length(test_ind[[user]])), length(k)), ncol = length(k))
```

```r
  # initialize user's prediction matrix
  # This has a column for each k, a row for each movie to be predicted

  # Create a smaller film ratings data table from our scaled ratings table. This cuts
  # out thousands of observations and contains only those ratings that may help us with
  # predictions for the current user
  # This data set also contains the proximity rankings of other users, so we can quickly
  # figure out who the nearest neighbors are
  film_ratings <- ratings_train[userId != user & movieId %in% test_ind[[user]]]
  film_ratings <- setkey(merge(film_ratings, sim_rank, by = "userId"), "rank")
  n_film = 1
  for(film in test_ind[[user]]){ # for each film we are trying to predict the user's
    # rating for:
    current_film_dat <- film_ratings[movieId == film] # only look at obs with that film
    neighbors = sapply(k, function(x) min(x, nrow(current_film_dat))) # we will take the
    # knn if we can.
    # otherwise we look at however many users rated the film. Create vector of how many
    # we take per k
    prediction = sapply(neighbors, function(x) mean(current_film_dat[[5]][1:x]))
    preds[n_film,] = prediction # the nth row of the user's prediction matrix contains
    # predictions using
    # different values of k
    n_film = n_film + 1 # Go through the loop again looking at the next film
  }
  preds <- ifelse(is.na(preds), 0, preds) # If somehow there are any NA ratings, set
  # them equal to 0
  xbar = as.integer(means[means$userId == user, 2])
  xbar_mat = matrix(rep(xbar, nrow(preds)*ncol(preds)), nrow = nrow(preds))
  preds <- preds + xbar_mat # De-scale the ratings
  y_mat <- matrix(rep(data[user,test_ind[[user]]], length(k)), ncol = length(k))
  # Compare predictions to observed
  resid <- preds - y_mat
  SSE <- SSE + apply(resid, 2, function(x) sum(x^2))
}
```

```
## [1] "USER 1000: 127.274 seconds elapsed"
## [1] "USER 2000: 251.682 seconds elapsed"
## [1] "USER 3000: 354.063 seconds elapsed"
## [1] "USER 4000: 456.371 seconds elapsed"
## [1] "USER 5000: 557.22 seconds elapsed"
## [1] "USER 6000: 657.138 seconds elapsed"
```

```r
time_elapsed = (proc.time() - ptm)[[3]]
print(paste0("All ratings calculated. A total of ", as.character(time_elapsed), " seconds have passed"))
```

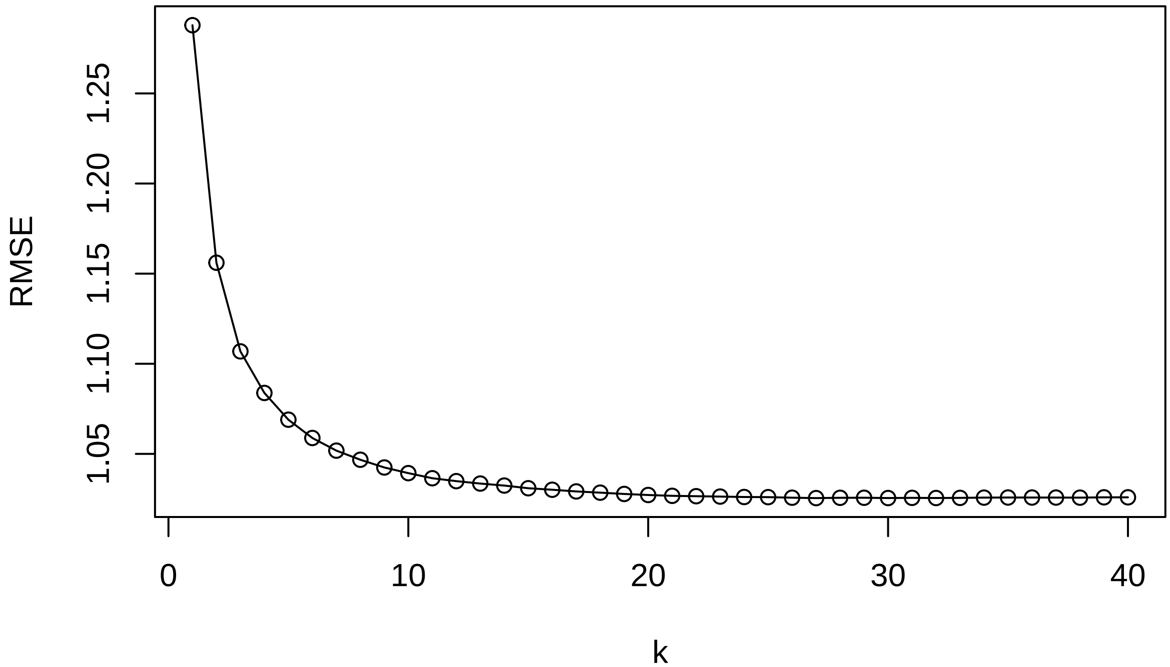
```
## [1] "All ratings calculated. A total of 661.604 seconds have passed"
```

Now let's look at how our predictor did, and which value of k is the best:

```r
# Get total number of movies we predicted so we can calculate MSE
inds <- lapply(test_ind, function(x) length(x))
testN = 0
for(user in inds){
  testN = testN + user
}
```

```r
# Get MSE, RMSE
MSE <- SSE/testN
RMSE <- sqrt(MSE)

plot(k, RMSE, type = "o")
```




```r
which.min(RMSE)
```

```
## [1] 27
```

```r
min(RMSE)
```

```
## [1] 1.025474
```

Now let's try SVD:

```r
n_vec = 100
svdM <- irlba(train_scaled, nv = n_vec) # Use train data, not scaled data

names(means)[2] <- "mean"
means <- as.data.frame(means)

error_vec = rep(0, 20)
j = 1

ptm = proc.time()

for(n in seq(5, 100, 5)){
  U <- svdM$u[,1:n]
  V = svdM$v[,1:n]
  D = svdM$d[1:n]
  # https://www.youtube.com/watch?v=-2pyabMzAto
  # You can compute the estimated rating of movie j by user i by computing U^T_i V_j
  SSE <- 0
  for(user in 1:length(test_ind)){
```

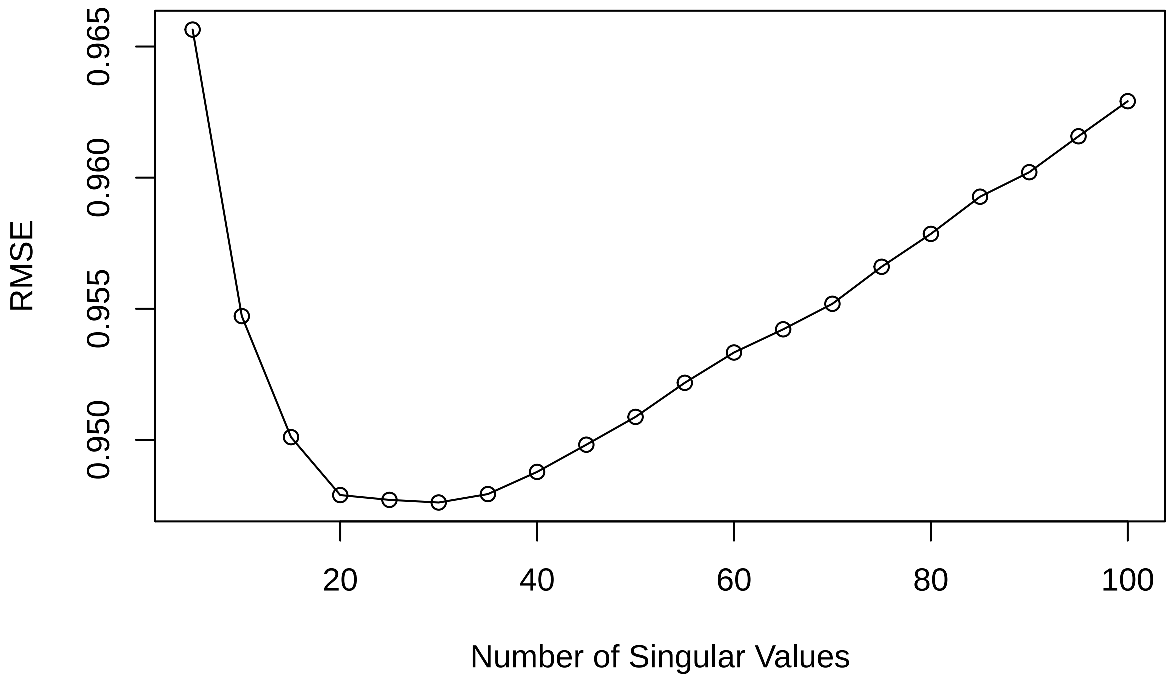
```r
    #if(user %% 100 == 0){print(user)}
    i = 1
    pred_vec = rep(0, length(test_ind[[user]])) # Empty vector to store predictions for current user
    for(film in test_ind[[user]]){ # Go thru each film
      pred_vec[i] <- t(U[user,])%*%diag(D)%*%(V[film,]) # Get prediction
      i <- i + 1 # Move to next movie
    }
    y <- data[user, test_ind[[user]]] # Get actual values
    pred_vec = pred_vec + means[user, 2]
    SSE <- SSE + sum((y - pred_vec)^2)
  }
  error_vec[j] <- SSE
  j <- j + 1
  print(paste0("SVD with ", as.character(n), " components complete: ", (proc.time() - ptm)[[3]], " secon
}
```

```
## [1] "SVD with 5 components complete: 16.3440000000001 seconds have elapsed"
## [1] "SVD with 10 components complete: 33.4250000000002 seconds have elapsed"
## [1] "SVD with 15 components complete: 49.788 seconds have elapsed"
## [1] "SVD with 20 components complete: 65.7450000000001 seconds have elapsed"
## [1] "SVD with 25 components complete: 81.5440000000001 seconds have elapsed"
## [1] "SVD with 30 components complete: 99.2370000000001 seconds have elapsed"
## [1] "SVD with 35 components complete: 115.548 seconds have elapsed"
## [1] "SVD with 40 components complete: 131.995 seconds have elapsed"
## [1] "SVD with 45 components complete: 149.904 seconds have elapsed"
## [1] "SVD with 50 components complete: 168.736 seconds have elapsed"
## [1] "SVD with 55 components complete: 188.731 seconds have elapsed"
## [1] "SVD with 60 components complete: 207.632 seconds have elapsed"
## [1] "SVD with 65 components complete: 226.572 seconds have elapsed"
## [1] "SVD with 70 components complete: 246.904 seconds have elapsed"
## [1] "SVD with 75 components complete: 267.228 seconds have elapsed"
## [1] "SVD with 80 components complete: 289.342 seconds have elapsed"
## [1] "SVD with 85 components complete: 310.991 seconds have elapsed"
## [1] "SVD with 90 components complete: 336.213 seconds have elapsed"
## [1] "SVD with 95 components complete: 359.932 seconds have elapsed"
## [1] "SVD with 100 components complete: 384.136 seconds have elapsed"
```

```r
print(paste0("All SVD loops ", as.character(n), " are complete: ", (proc.time() - ptm)[[3]], " seconds h
```

```
## [1] "All SVD loops 100 are complete: 384.138 seconds have elapsed"
```

```r
plot(seq(5,100,5), sqrt(error_vec/testN), type = "o", xlab = "Number of Singular Values", ylab = "RMSE")
```




```r
seq(5,100,5)[which.min(error_vec)]
```

```
## [1] 30
```

```r
(error_vec/testN)[which.min(error_vec)]
```

```
## [1] 0.8979625
```

These results seem consistent with previous experiments with recommender systems (KNN gives us an RMSE of around 1, and SVD gives about .9. See http://cs229.stanford.edu/proj2012/BaoXia-MovieRatingEstimationAndRecommendation_FinalWriteup.pdf)

In addition to being more accurate, our predictions using SVD can be obtained much more quickly than those with KNN.