ETC3555 2018 - Lab 9 solutions

Recommender Systems

Cameron Roach and Souhaib Ben Taieb

11 October, 2018

In this assignment, you will implement the collaborative filtering learning algorithm and apply it to a dataset of movie ratings. This dataset consists of ratings on a scale of 1 to 5. The dataset has $n_u = 943$ users, and $n_m = 1682$ movies.

```
load("movies_ratings.Rda")
list2env(data,.GlobalEnv) # "Y" and "R"
```

```
## <environment: R_GlobalEnv>
```

```
rm(data)

#  Y is a 1682x943 matrix, containing ratings (1-5) of 1682 movies on
#  943 users
#
#  R is a 1682x943 matrix, where R[i,j] <- 1 if and only if user j gave a
#  rating to movie i
```

The following code gives all movies titles.

```
source("loadMovieList.R")
movieList <- loadMovieList()
```

(1 mark) Compute the average ratings for "Toy Story (1995)" and "Alaska (1996)".

```
# ----------------------- YOUR CODE HERE -----------------------
library(tidyverse)

ifelse(R, Y, NA) %>%
  t() %>%
  magrittr::set_colnames(movieList) %>%
  as_data_frame() %>%
  select("Toy Story (1995)", "Alaska (1996)") %>%
  summarise_all(funs(mean(., na.rm=TRUE)))
```

```
## # A tibble: 1 x 2
##   `Toy Story (1995)` `Alaska (1996)`
##                <dbl>           <dbl>
## 1               3.88            2.69
```

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

(1 mark) Which user has rated more movies? Plot a histogram of her/his ratings?

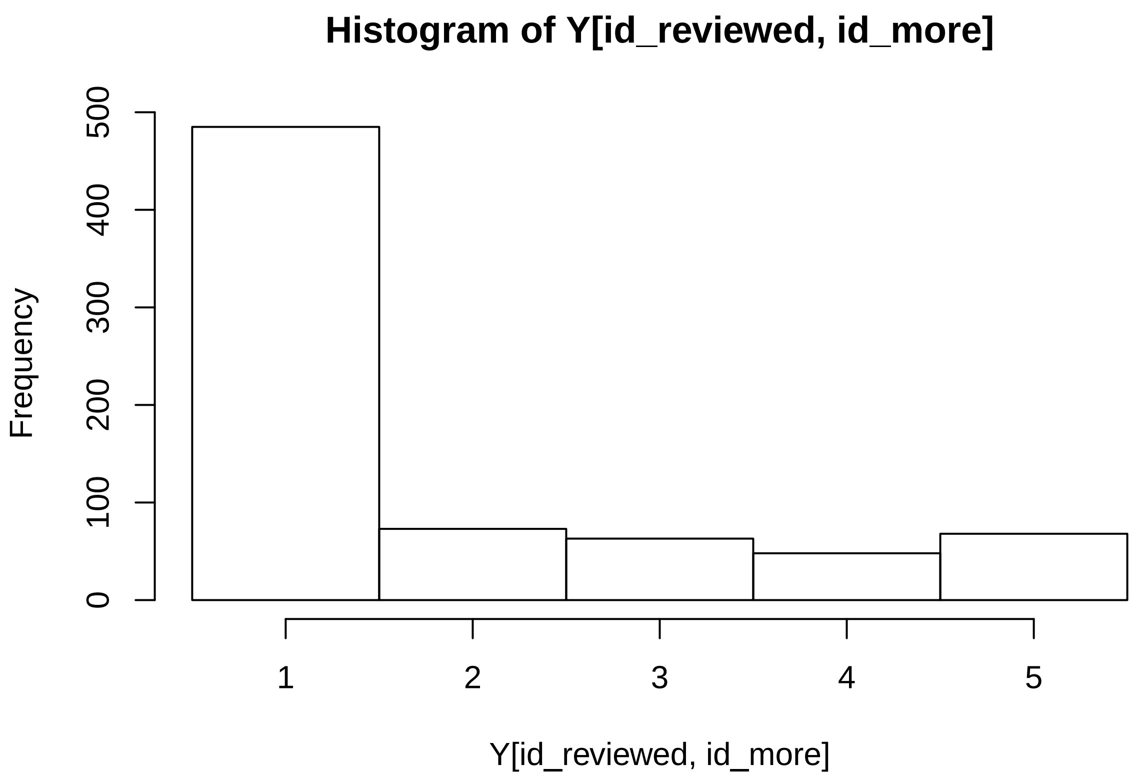
```
# ----------------------- YOUR CODE HERE -----------------------
id_more <- which.max(apply(R, 2, sum))
id_reviewed <- which(R[, id_more] == 1)
print(id_more)
```

```
## [1] 405
```

```r
print(length(Y[id_reviewed, id_more]))
```

```
## [1] 737
```

```r
hist(Y[id_reviewed, id_more], .5+0:5)
```




```r
# -------------------------------------------------------------
```

You will now implement the cost function for collaborative filtering. Specifically, you should complete the following code to return J.

```r
cofiCostFunc <- function(Y, R, num_users, num_movies,
                         num_features, lambda = 0) {
  #COFICOSTFUNC Collaborative filtering cost function
  #   J <- COFICOSTFUNC(Y, R, num_users, num_movies, ...
  #   num_features, lambda)(params) returns the cost for the
  #   collaborative filtering problem.
  #
  # Notes: X - num_movies  x num_features matrix of movie features
  #        Theta - num_users  x num_features matrix of user features
  #        Y - num_movies x num_users matrix of user ratings of movies
  #        R - num_movies x num_users matrix, where R(i, j) <- 1 if the
  #            i-th movie was rated by the j-th user

  function(params) {
    # Unfold the U and W matrices from params
    X <-
      matrix(params[1:(num_movies * num_features)], num_movies, num_features)
    Theta <-
      matrix(params[(num_movies * num_features + 1):length(params)],num_users, num_features)
```

```r
    # ----------------------- YOUR CODE HERE -----------------------
    J <- 0
    J <- (1 / 2) * sum(((X %*% t(Theta)) * R - Y * R) ^ 2) +
      (lambda / 2 * sum(Theta ^ 2)) + (lambda / 2 * sum(X ^ 2))
    # --------------------------------------------------------------
    J
  }
}
```

To help you debug your cost function, run the following code.

```r
#  Load pre-trained weights (X, Theta, num.users, num.movies, num.features)
load("movieParams.Rda")
list2env(data,.GlobalEnv)
```

```
## <environment: R_GlobalEnv>
```

```r
rm(data)

num_users <- as.numeric(num.users)
num_movies <- as.numeric(num.movies)
num_features <- as.numeric(num.features)

#  Reduce the data set size so that this runs faster
num_users <- 4; num_movies <- 5; num_features <- 3
X <- X[1:num_movies, 1:num_features]
Theta <- Theta[1:num_users, 1:num_features]
Y <- Y[1:num_movies, 1:num_users]
R <- R[1:num_movies, 1:num_users]

#  Evaluate cost function
J <- cofiCostFunc(Y, R, num_users, num_movies, num_features, 0)(c(c(X),c(Theta)))

cat(sprintf('Cost at loaded parameters: %f  (this value should be about 22.22)\n', J))
```

```
## Cost at loaded parameters: 22.224604  (this value should be about 22.22)
```

Once your cost function matches up with ours, you should now implement the collaborative filtering gradient function. Specifically, you should complete the following code to return the grad argument.

```r
cofiGradFunc <- function(Y, R, num_users, num_movies, num_features, lambda = 0) {
  #cofiGradFunc returns the gradient for the
  #   collaborative filtering problem.
  # Notes: X - num_movies  x num_features matrix of movie features
  #        Theta - num_users  x num_features matrix of user features
  #        Y - num_movies x num_users matrix of user ratings of movies
  #        R - num_movies x num_users matrix, where R[i, j] <- 1 if the
  #            i-th movie was rated by the j-th user
  #
  function(params) {
    # Unfold the U and W matrices from params
    X <-
      matrix(params[1:(num_movies * num_features)], num_movies, num_features)
    Theta <-
      matrix(params[(num_movies * num_features + 1):length(params)],
             num_users, num_features)
```

```r
    # You should set the following variables correctly:
    #        X_grad - num_movies x num_features matrix, containing the
    #                 partial derivatives w.r.t. to each element of X
    #        Theta_grad - num_users x num_features matrix, containing the
    #                     partial derivatives w.r.t. to each element of Theta
    #

    # ----------------------- YOUR CODE HERE -----------------------
    X_grad <- matrix(0,dim(X)[1],dim(X)[2])
    Theta_grad <- matrix(0, dim(Theta)[1], dim(Theta)[2])

    X_grad <- (((X %*% t(Theta)) * R) %*% Theta - (Y * R) %*% Theta) + lambda * X
    Theta_grad <- t((t(X) %*% ((X %*% t(Theta)) * R) - t(X) %*% (Y * R))) + lambda * Theta
    # ---------------------------------------------------------------
    grad <- c(c(X_grad),c(Theta_grad))
    grad
  }
}
```

You can check your function by running the following code.

```r
source("computeNumericalGradient.R")
source("checkCostFunction.R")
checkCostFunction()
```

```
##           numgrad        grad
##  [1,] -0.55499681 -0.55499681
##  [2,]  0.69251864  0.69251864
##  [3,] -2.42820624 -2.42820624
##  [4,] -2.86158362 -2.86158362
##  [5,]  4.36825009  4.36825009
##  [6,]  2.13383447  2.13383447
##  [7,]  2.06852264  2.06852264
##  [8,]  0.03181708  0.03181708
##  [9,]  2.43105156  2.43105156
## [10,] -2.80455561 -2.80455561
## [11,]  3.89199336  3.89199336
## [12,] -0.18570459 -0.18570459
## [13,] -0.56876254 -0.56876254
## [14,] -0.82111580 -0.82111580
## [15,]  3.48401697  3.48401697
## [16,] -1.36020376 -1.36020376
## [17,]  0.44965466  0.44965466
## [18,] -0.35655886 -0.35655886
## [19,] -1.84466173 -1.84466173
## [20,]  0.89889296  0.89889296
## [21,] -3.89654172 -3.89654172
## [22,]  2.34042388  2.34042388
## [23,] -0.73259714 -0.73259714
## [24,]  0.50044757  0.50044757
## [25,]  1.48144603  1.48144603
## [26,] -0.10139936 -0.10139936
## [27,] -0.84233864 -0.84233864
```

```
## The above two columns you get should be very similar.
##     (Left-Your Numerical Gradient, Right-Analytical Gradient)
##
## If your implementation is correct, then
##       the relative difference will be small (less than 1e-9).
##
##       Relative Difference: 1.08708e-12
```

(3 marks) Now, you should implement regularization for the cost function for collaborative filtering. Update the code of the *cofiCostFunc* function.

You can check your function using the following code.

```r
lambda <- 1.5
J <- cofiCostFunc(Y, R, num_users, num_movies, num_features, lambda)(c(c(X),c(Theta)))

cat(sprintf('Cost at loaded parameters (lambda = 1.5): %f (this value should be about 31.34)\n', J))
```

```
## Cost at loaded parameters (lambda = 1.5): 31.344056 (this value should be about 31.34)
```

(3 marks) Once your cost matches up with ours, you should proceed to implement regularization for the gradient. Update the code of the *cofiGradFunc* function.

You can check the gradient computations with the following code.

```r
checkCostFunction(lambda)
```

```
##           numgrad        grad
##  [1,] -4.43276533 -4.43276533
##  [2,] -1.86651963 -1.86651963
##  [3,]  8.90525711  8.90525711
##  [4,] -4.64843021 -4.64843021
##  [5,] -2.71215493 -2.71215493
##  [6,]  4.02522675  4.02522675
##  [7,] -1.87156418 -1.87156418
##  [8,]  4.70344519  4.70344519
##  [9,] -2.46958225 -2.46958225
## [10,] -4.29884118 -4.29884118
## [11,]  6.14974070  6.14974070
## [12,]  1.48294980  1.48294980
## [13,] -2.97603592 -2.97603592
## [14,] -5.11548970 -5.11548970
## [15,] -2.86991467 -2.86991467
## [16,]  2.35779046  2.35779046
## [17,]  4.14254977  4.14254977
## [18,]  2.90230180  2.90230180
## [19,]  0.31841957  0.31841957
## [20,] -3.69962010 -3.69962009
## [21,] -0.04912758 -0.04912758
## [22,] -3.34423305 -3.34423305
## [23,] -3.36219218 -3.36219218
## [24,] -4.72380997 -4.72380997
## [25,] -0.03017356 -0.03017356
## [26,]  2.73996071  2.73996071
## [27,] -1.43443899 -1.43443899
## The above two columns you get should be very similar.
##     (Left-Your Numerical Gradient, Right-Analytical Gradient)
##
```

```
## If your implementation is correct, then
##       the relative difference will be small (less than 1e-9).
##
##       Relative Difference: 2.45535e-12
```

Before training the collaborative filtering model, we will first add ratings that correspond to a new user that we just observed. This part of the code will allow you to put in your own ratings for the movies in our dataset!

```r
#  Initialize my ratings
my_ratings <- rep(0,1682)

# Check the file movie_ids.txt for id of each movie in our dataset
# For example, Toy Story (1995) has ID 1, so to rate it "4", you can set
my_ratings[1] <- 4

# Or suppose did not enjoy Silence of the Lambs (1991), you can set
my_ratings[98] <- 2

# We have selected a few movies we liked / did not like and the ratings we
# gave are as follows:
my_ratings[7] <- 3
my_ratings[12]<- 5
my_ratings[54] <- 4
my_ratings[64]<- 5
my_ratings[66]<- 3
my_ratings[69] <- 5
my_ratings[183] <- 4
my_ratings[226] <- 5
my_ratings[355]<- 5

cat(sprintf('\n\nNew user ratings:\n'))
```

```
##
##
## New user ratings:
```

```r
for (i in 1:length(my_ratings))
    if (my_ratings[i] > 0 )
        cat(sprintf('Rated %d for %s\n', my_ratings[i], movieList[i]))
```

```
## Rated 4 for Toy Story (1995)
## Rated 3 for Twelve Monkeys (1995)
## Rated 5 for Usual Suspects, The (1995)
## Rated 4 for Outbreak (1995)
## Rated 5 for Shawshank Redemption, The (1994)
## Rated 3 for While You Were Sleeping (1995)
## Rated 5 for Forrest Gump (1994)
## Rated 2 for Silence of the Lambs, The (1991)
## Rated 4 for Alien (1979)
## Rated 5 for Die Hard 2 (1990)
## Rated 5 for Sphere (1998)
```

Now we will train the collaborative filtering model.

```r
load("movies_ratings.Rda")
list2env(data,.GlobalEnv) # "Y" and "R"
```

```
## <environment: R_GlobalEnv>
```

```r
rm(data)

#  Add our own ratings to the data matrix

Y <- cbind(my_ratings, Y)
R <- cbind((my_ratings != 0), R)

#  Normalize Ratings
source("normalizeRatings.R")
NR  <- normalizeRatings(Y, R)
Ynorm <- NR$Ynorm
Ymean <- NR$Ymean
#  Useful Values
num_users <- dim(Y)[2]
num_movies <- dim(Y)[1]
num_features <- 10

# Set Initial Parameters (Theta, X)
n <- num_movies * num_features
X <- matrix(rnorm(n), num_movies, num_features)

n <- num_users * num_features
Theta <-  matrix(rnorm(n), num_users, num_features)

initial_parameters <- c(c(X), c(Theta))

# Set Regularization
lambda <- 10

cF <- cofiCostFunc(Ynorm, R, num_users, num_movies,num_features, lambda)
gF <- cofiGradFunc(Ynorm, R, num_users, num_movies,num_features, lambda)

#install.packages("lbfgsb3")
library(lbfgsb3)
source("lbfgsb3_.R")
theta <- lbfgsb3_(initial_parameters, fn= cF, gr=gF, control = list(trace=1,maxit=100))$prm
```

```
## This problem is unconstrained.
## At iteration  0  f = 691521.2
## At iteration  2  f = 670660.8
## At iteration  3  f = 593480.1
## At iteration  4  f = 238705.1
## At iteration  5  f = 157828.1
## At iteration  6  f = 110506.8
## At iteration  7  f = 86787.43
## At iteration  8  f = 62428.85
## At iteration  9  f = 56515.35
## At iteration  10  f = 50350.14
## At iteration  11  f = 45894.19
## At iteration  12  f = 44464.24
## At iteration  13  f = 42849.04
## At iteration  14  f = 41895.81
## At iteration  15  f = 41350.4
```

```
## At iteration  16  f = 40771.77
## At iteration  17  f = 40396.03
## At iteration  18  f = 40085.82
## At iteration  19  f = 39881.12
## At iteration  20  f = 39742.31
## At iteration  21  f = 39584.01
## At iteration  22  f = 39528.39
## At iteration  23  f = 39436.02
## At iteration  24  f = 39386.29
## At iteration  25  f = 39326.8
## At iteration  26  f = 39273.87
## At iteration  27  f = 39232.24
## At iteration  28  f = 39205.37
## At iteration  29  f = 39178.56
## At iteration  30  f = 39152.69
## At iteration  31  f = 39133.48
## At iteration  32  f = 39119.98
## At iteration  33  f = 39096.5
## At iteration  34  f = 39085.85
## At iteration  35  f = 39072.25
## At iteration  36  f = 39059.13
## At iteration  37  f = 39048.92
## At iteration  38  f = 39043.53
## At iteration  39  f = 39028.1
## At iteration  40  f = 39022.55
## At iteration  41  f = 39012.83
## At iteration  42  f = 39009.09
## At iteration  43  f = 39001.61
## At iteration  44  f = 38998.25
## At iteration  45  f = 38994.79
## At iteration  46  f = 38991.11
## At iteration  47  f = 38987.68
## At iteration  48  f = 38986.26
## At iteration  49  f = 38984.48
## At iteration  50  f = 38982.63
## At iteration  51  f = 38981.36
## At iteration  52  f = 38980.68
## At iteration  53  f = 38979.86
## At iteration  54  f = 38979.16
## At iteration  55  f = 38978.45
## At iteration  56  f = 38978
## At iteration  57  f = 38977.54
## At iteration  58  f = 38977.21
## At iteration  59  f = 38976.86
## At iteration  60  f = 38976.45
## At iteration  61  f = 38976.24
## At iteration  62  f = 38976.02
## At iteration  63  f = 38975.76
## At iteration  64  f = 38975.52
## At iteration  65  f = 38975.32
## At iteration  66  f = 38974.91
## At iteration  67  f = 38974.72
## At iteration  68  f = 38974.41
## At iteration  69  f = 38974.01
```

```
## At iteration  70  f = 38973.57
## At iteration  71  f = 38973.2
## At iteration  72  f = 38972.71
## At iteration  73  f = 38972.46
## At iteration  74  f = 38972.15
## At iteration  75  f = 38971.74
## At iteration  76  f = 38971.46
## At iteration  77  f = 38971.15
## At iteration  78  f = 38970.57
## At iteration  79  f = 38970.31
## At iteration  80  f = 38969.93
## At iteration  81  f = 38969.85
## At iteration  82  f = 38969.28
## At iteration  83  f = 38969.08
## At iteration  84  f = 38968.73
## At iteration  85  f = 38968.22
## At iteration  86  f = 38968.47
## At iteration  87  f = 38967.89
## At iteration  88  f = 38967.39
## At iteration  89  f = 38966.98
## At iteration  90  f = 38966.54
## At iteration  91  f = 38966.16
## At iteration  92  f = 38965.68
## At iteration  93  f = 38965.17
## At iteration  94  f = 38964.75
## At iteration  95  f = 38964.36
## At iteration  96  f = 38963.8
## At iteration  97  f = 38963.26
## At iteration  98  f = 38962.78
## At iteration  99  f = 38962.13
## At iteration  100  f = 38961.38
## At iteration  101  f = 38960.87
```

```r
# The following code works but optim is slow on this problem
# theta <- optim(initial_parameters, fn = cF, gr = gF,
       #method = "BFGS", control = list(maxit=10, trace=1, REPORT=1) )$par

# Unfold the returned theta back into U and W
X <- matrix(theta[1:(num_movies*num_features)], num_movies, num_features)
Theta <- matrix(theta[(num_movies*num_features+1):length(theta)], num_users, num_features)

cat(sprintf('Recommender system learning completed.\n'))
```

```
## Recommender system learning completed.
```

(2 marks) After training the model, you have now computed X and Theta. Use them to compute the top 10 recommendations for the new user. Print the movie titles and the associated ratings.

```r
# ------------------------ YOUR CODE HERE ------------------------
p <- X %*% t(Theta)
my_predictions <- p[,1] + Ymean

movieList <- loadMovieList()

ix <- sort(my_predictions, decreasing = TRUE,index.return=TRUE)$ix
```

```r
cat(sprintf('\nTop recommendations for you:\n'))
```

```
##
## Top recommendations for you:
```

```r
for (i in 1:10){
    j <- ix[i]
    cat(sprintf('Predicting rating %.1f for movie %s\n', my_predictions[j],movieList[j]))
}
```

```
## Predicting rating 5.0 for movie Great Day in Harlem, A (1994)
## Predicting rating 5.0 for movie They Made Me a Criminal (1939)
## Predicting rating 5.0 for movie Santa with Muscles (1996)
## Predicting rating 5.0 for movie Saint of Fort Washington, The (1993)
## Predicting rating 5.0 for movie Prefontaine (1997)
## Predicting rating 5.0 for movie Star Kid (1997)
## Predicting rating 5.0 for movie Entertaining Angels: The Dorothy Day Story (1996)
## Predicting rating 5.0 for movie Aiqing wansui (1994)
## Predicting rating 5.0 for movie Marlene Dietrich: Shadow and Light (1996)
## Predicting rating 5.0 for movie Someone Else's America (1995)
```

```r
cat(sprintf('\n\nOriginal ratings provided:\n'))
```

```
##
##
## Original ratings provided:
```

```r
for (i in 1:length(my_ratings))
    if (my_ratings[i] > 0 )
        cat(sprintf('Rated %d for %s\n', my_ratings[i],movieList[i]))
```

```
## Rated 4 for Toy Story (1995)
## Rated 3 for Twelve Monkeys (1995)
## Rated 5 for Usual Suspects, The (1995)
## Rated 4 for Outbreak (1995)
## Rated 5 for Shawshank Redemption, The (1994)
## Rated 3 for While You Were Sleeping (1995)
## Rated 5 for Forrest Gump (1994)
## Rated 2 for Silence of the Lambs, The (1991)
## Rated 4 for Alien (1979)
## Rated 5 for Die Hard 2 (1990)
## Rated 5 for Sphere (1998)
```

```r
# -----------------------------------------------------------------
```

TURN IN

- Your `.Rmd` file (which should knit without errors and without assuming any packages have been pre-loaded)
- Your Word (or pdf) file that results from knitting the Rmd.
- DUE: October 2, 11:55pm (late submissions not allowed), loaded into moodle.