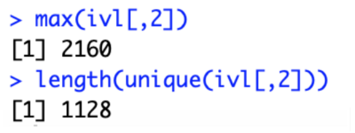
ECS189G Homework 2 Report Problem B

Goh Chang Kang, Charles 916751838,
Yang Minxing 916751773, Chen Jieyi Chloe 999823783

October 30, 2018

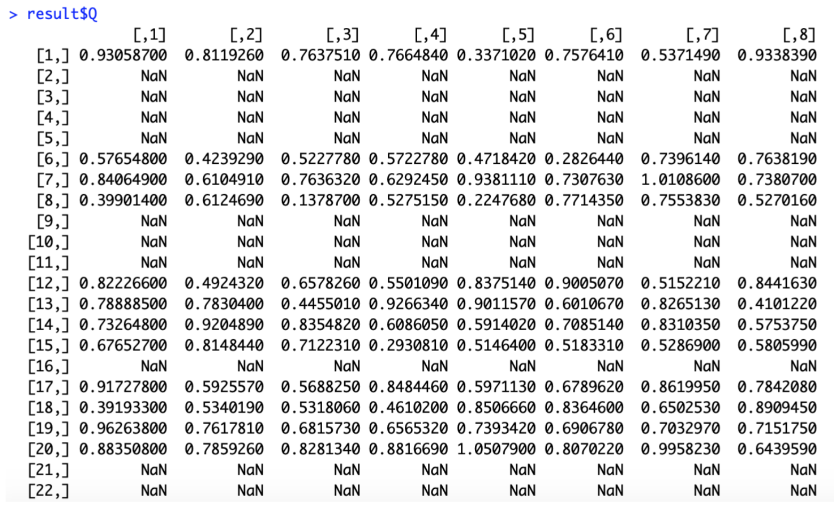
1 Background

In the ivl data set, the user and item ID indicates the identity of a user and a professor. We cannot assume that they are consecutive because there may be some user or item ids that are missing. For example I could have user ids 1, 2, and 4. This would mean I am missing id 3. This is apparent in the dataset when we check the maximum item ID vs the number of unique IDs



```
> max(ivl[,2])
[1] 2160
> length(unique(ivl[,2]))
[1] 1128
```

We can also see this during the matrix factorisation process. When the matrix is split into its component vectors P and Q, the former representing users and the latter representing items, we see that actually the Q matrix is not full rank and contains many NaN values. This will cause problems when multiplying, and would also cause problems during computation since in R any operation on NaN returns a NaN value



```
> result$Q
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]
 [1,] 0.93058700 0.8119260 0.7637510 0.7664840 0.3371020 0.7576410 0.5371490 0.9338390
 [2,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
 [3,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
 [4,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
 [5,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
 [6,] 0.57654800 0.4239290 0.5227780 0.5722780 0.4718420 0.2826440 0.7396140 0.7638190
 [7,] 0.84064900 0.6104910 0.7636320 0.6292450 0.9381110 0.7307630 1.0108600 0.7380700
 [8,] 0.39901400 0.6124690 0.1378700 0.5275150 0.2247680 0.7714350 0.7553830 0.5270160
 [9,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
[10,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
[11,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
[12,] 0.82226600 0.4924320 0.6578260 0.5501090 0.8375140 0.9005070 0.5152210 0.8441630
[13,] 0.78888500 0.7830400 0.4455010 0.9266340 0.9011570 0.6010670 0.8265130 0.4101220
[14,] 0.73264800 0.9204890 0.8354820 0.6086050 0.5914020 0.7085140 0.8310350 0.5753750
[15,] 0.67652700 0.8148440 0.7122310 0.2930810 0.5146400 0.5183310 0.5286900 0.5805990
[16,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
[17,] 0.91727800 0.5925570 0.5688250 0.8484460 0.5971130 0.6789620 0.8619950 0.7842080
[18,] 0.39193300 0.5340190 0.5318060 0.4610200 0.8506660 0.8364600 0.6502530 0.8909450
[19,] 0.96263800 0.7617810 0.6815730 0.6565320 0.7393420 0.6906780 0.7032970 0.7151750
[20,] 0.88350800 0.7859260 0.8281340 0.8816690 1.0507900 0.8070220 0.9958230 0.6439590
[21,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
[22,]        NaN       NaN       NaN       NaN       NaN       NaN       NaN       NaN
```

To solve this problem, we made changes to the trainReco() function as well as the predict() function

2 Changes to trainReco()

For the purposes of simplicity, the changes for the function when dealing with item id will be described. The same changes are made when dealing with user id. For item id, we take the maximum id and the number of unique ids. If they are equal, we do not need to rectify it. If they are not equal, that means there are non-consecutive item ids. We then compress the ids into consecutive ones without changing the relative position in the table. Finally at the end of the function we added a translator dataframe to our output result. This will serve as the translator of original id to translated id in the predict function.

```
library(rectools)
trainReco <- function (ratingsIn, rnk = 10, nmf = FALSE)
{
  require(recosystem)
  hasCovs <- (ncol(ratingsIn) > 3)
  if (hasCovs) {
    covs <- as.matrix(ratingsIn[, -(1:3)])
    lmout <- lm(ratingsIn[, 3] ~ covs)
    minResid <- min(lmout$residuals)
    ratingsIn[, 3] <- lmout$residuals - minResid
  }
  #check for consecutive records.
  #if the max ID is greater than the num of rows, means they are not consecutive
  sMax <- max(ratingsIn[, 1])
  sUnique <- length(unique(ratingsIn[, 1]))
  sOriginal <- ratingsIn[, 1]

  dMax <- max(ratingsIn[, 2])
  dUnique <- length(unique(ratingsIn[, 2]))
  dOriginal <- ratingsIn[, 2]

  #if they are not consecutive, we use the shrinked ones
  if ((sMax - sUnique) > 0) {
    sortedIdx <- sort(unique(ratingsIn[, 1]))
    sIDTemp <- match(sOriginal, sortedIdx)
    #use the relative position as the new ID, because positions must be consecutive
  } else {
    sIDTemp <- sOriginal
  }

  if ((dMax - dUnique) > 0) {
    sortedIdx <- sort(unique(ratingsIn[, 2]))
    dIDTemp <- match(dOriginal, sortedIdx)
  } else {
    dIDTemp <-dOriginal
  }

  r <- Reco()
  train_set <- data_memory(sIDTemp, dIDTemp,
                           ratingsIn[, 3], index1 = TRUE)
  r$train(train_set, opts = list(dim = rnk, nmf = nmf))
  result <- r$output(out_memory(), out_memory())
  attr(result, "hasCovs") <- hasCovs
  if (hasCovs) {
    attr(result, "covCoefs") <- coef(lmout)
    attr(result, "minResid") <- minResid
  }
  class(result) <- "RecoS3"
  #append the attribute for translation purpose
  result$translator <- data.frame(sIDTemp, dIDTemp, sOriginal, dOriginal)
  result
}
```

3 Changes to predict.RecoS3()

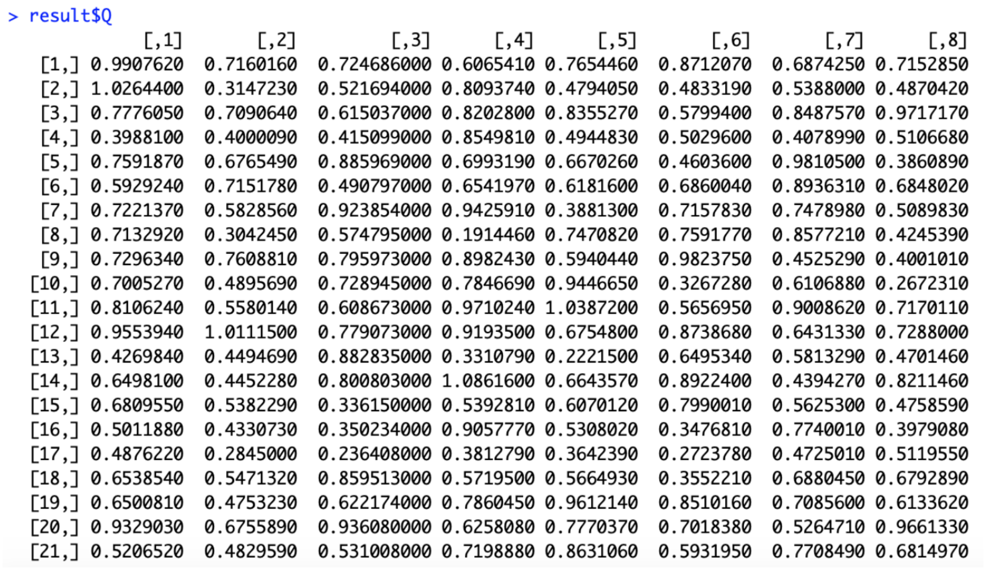
In the predict.RecoS3 function, we grab the appended translator dataframe and search for the translated id in the compressed version using the inputted id (which is the original reference).

```
predict.RecoS3 <- function (recoObj, testSet)
{
  p <- recoObj$P
  q <- recoObj$Q
  testSet$pred <- vector(length = nrow(testSet))
  hasCovs <- attr(recoObj, "hasCovs")
  if (hasCovs) {
    covCoefs <- attr(recoObj, "covCoefs")
    minResid <- attr(recoObj, "minResid")
  }
  for (i in 1:nrow(testSet)) {
    j <- testSet[i, 1]
    k <- testSet[i, 2]
    #use the translator to translate to consecutive ID
    originalSIndex <- match(j, result$translator[[3]])
    originalDIndex <- match(k, result$translator[[4]])
    j <- result$translator[[1]][originalSIndex]
    k <- result$translator[[2]][originalDIndex]
    if (!is.na(j) && !is.na(k) && j <= nrow(p) && k <= nrow(q)) {
      tmp <- 0
      if (hasCovs)
        tmp <- covCoefs %*% c(1, testSet[i, -(1:3)]) -
          minResid
      testSet$pred[i] <- p[j, ] %*% q[k, ] + tmp
    }
    else testSet$pred[i] <- NA
  }
  testSet$pred
}
```

Note: We also added an extra check for non-existing values when predicting (as quoted below):

```
if (!is.na(j) && !is.na(k) && j <= nrow(p) && k <= nrow(q))
```

After fixing the non-consecutive id problem, we can re-check our matrix-factorisation components and see that there are no more NaN values, thus confirming that we have indeed compressed the ids successfully.



```
> result$Q
           [,1]      [,2]        [,3]      [,4]      [,5]      [,6]      [,7]      [,8]
 [1,] 0.9907620 0.7160160 0.724686000 0.6065410 0.7654460 0.8712070 0.6874250 0.7152850
 [2,] 1.0264400 0.3147230 0.521694000 0.8093740 0.4794050 0.4833190 0.5388000 0.4870420
 [3,] 0.7776050 0.7090640 0.615037000 0.8202800 0.8355270 0.5799400 0.8487570 0.9717170
 [4,] 0.3988100 0.4000090 0.415099000 0.8549810 0.4944830 0.5029600 0.4078990 0.5106680
 [5,] 0.7591870 0.6765490 0.885969000 0.6993190 0.6670260 0.4603600 0.9810500 0.3860890
 [6,] 0.5929240 0.7151780 0.490797000 0.6541970 0.6181600 0.6860040 0.8936310 0.6848020
 [7,] 0.7221370 0.5828560 0.923854000 0.9425910 0.3881300 0.7157830 0.7478980 0.5089830
 [8,] 0.7132920 0.3042450 0.574795000 0.1914460 0.7470820 0.7591770 0.8577210 0.4245390
 [9,] 0.7296340 0.7608810 0.795973000 0.8982430 0.5940440 0.9823750 0.4525290 0.4001010
[10,] 0.7005270 0.4895690 0.728945000 0.7846690 0.9446650 0.3267280 0.6106880 0.2672310
[11,] 0.8106240 0.5580140 0.608673000 0.9710240 1.0387200 0.5656950 0.9008620 0.7170110
[12,] 0.9553940 1.0111500 0.779073000 0.9193500 0.6754800 0.8738680 0.6431330 0.7288000
[13,] 0.4269840 0.4494690 0.882835000 0.3310790 0.2221500 0.6495340 0.5813290 0.4701460
[14,] 0.6498100 0.4452280 0.800803000 1.0861600 0.6643570 0.8922400 0.4394270 0.8211460
[15,] 0.6809550 0.5382290 0.336150000 0.5392810 0.6070120 0.7990010 0.5625300 0.4758590
[16,] 0.5011880 0.4330730 0.350234000 0.9057770 0.5308020 0.3476810 0.7740010 0.3979080
[17,] 0.4876220 0.2845000 0.236408000 0.3812790 0.3642390 0.2723780 0.4725010 0.5119550
[18,] 0.6538540 0.5471320 0.859513000 0.5719500 0.5664930 0.3552210 0.6880450 0.6792890
[19,] 0.6500810 0.4753230 0.622174000 0.7860450 0.9612140 0.8510160 0.7085600 0.6133620
[20,] 0.9329030 0.6755890 0.936080000 0.6258080 0.7770370 0.7018380 0.5264710 0.9661330
[21,] 0.5206520 0.4829590 0.531008000 0.7198880 0.8631060 0.5931950 0.7708490 0.6814970
```