Advanced Data Science Project 4 Report

21/11/2019

Introduction

In this project, our group implement matrix factorization by focusing on different versions of alternating least squares algorithm for different Regularizations and KNN as Post-processing. The objective is to produce a good prediction of users' preferences for movies on on the Netflix Prize dataset. For this project, our team #2 is assigned with Pairing combination 9 + 11 + 13 from the Collaborative. For evaluation, we compared RMSE results for different methods. Our group used R lanauage to product model and reports. The specific technique used for the combination are illustrated below. You can also see (https://github.com/TZstatsADS/ADS_Teaching/blob/master/Projects_StarterCodes/Project4-RecommenderSystem/doc/Matrix%20Factorization.pdf) for more details of our methods.

Models

`Alternating Least Squares + KNN`

`Alternating Least Squares + Penalty of magnitudes + Bias and Intercepts + KNN`

`Alternating Least Squares + Temporal Dynamics + KNN`

Model 1:Alternating Least Squares + KNN

Equations recap

Alternating Least Squares

$$min_{q^\star p^\star} \sum_{(u,i)\in K} (r_u i - q_i^T p_u)^2 + \lambda(\sum_i n_{q_i}\|q_i\|^2 + \sum_i n_{p_u}\|p_u\|^2)$$

ALS technique rotate between fixing the q_i's and fixing the p_u's. When all pu's are fixed, system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each step decreases object function until convergence.

f: dimension of latent factors

q_i: factors associated with item i , measures the extent to which items possesses those factors

p_u: factors associated with user u, measures the extent of interest that user has in an item are high on corresponding factors.

Post-processing

KNN:

$$s(q_i, q_j) = \frac{q_i^T, q_j}{\|q_i\|\|q_j\|}$$

We first defined similarity between movies by the above formula. Then we used this similarity S to apply KNN prediction.

Codes and Details

Project 4 ALS (A3)

HANG HU hh2718

11/20/2019

Step 1 Load Data and Train-test Split

```
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
```

```
library(tidyr)
library(ggplot2)
data <- read.csv("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/data/ml-latest-small/ratings.csv")
set.seed(0)
## shuffle the row of the entire dataset
data <- data[sample(nrow(data)),]
## get a small dataset that contains all users and all movies
unique.user<-duplicated(data[,1])
unique.movie<-duplicated(data[,2])
index<-unique.user & unique.movie
all.user.movie <- data[!index,]

## split training and test on the rest
rest <- data[index,]
test_idx <- sample(rownames(rest), round(nrow(data)/5, 0))
train_idx <- setdiff(rownames(rest), test_idx)

## combine the training with the previous dataset, which has all users and all movies
data_train <- rbind(all.user.movie, data[train_idx,])
data_test <- data[test_idx,]

## sort the training and testing data by userId then by movieId,
## so when we update p and q, it is less likely to make mistakes
data_train <- arrange(data_train, userId, movieId)
data_test <- arrange(data_test, userId, movieId)
```

###Step 2 Matrix Factorization #### Step 2.1 Algorithm and Regularization Here I perform stochastic gradien descent to do matrix factorization. Your algorithm should consider case that there are new users and movies adding to the dataset you used to train. In other words, the dimension your matrix R, q, p is dynamic.

- For algorithms, the referenced paper are:

A1. Stochastic Gradient Descent Section: Learning Algorithms-Stochastic Gradient Descent

A2. Gradient Descent with Probabilistic Assumptions Section 2

A3. Alternating Least Squares Section 3.1

- For regularizations, the referenced paper are:

R1. Penalty of Magnitudes Section: a Basic Matrix Factorization Model

R2. Bias and Intercepts Section: Adding Biases

R3. Temporal Dynamics Section 4

```
U <- length(unique(data$userId))
I <- length(unique(data$movieId))
source("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/lib/ALS1.R")
```

Step 2.2 Parameter Tuning

Here you should tune parameters, such as the dimension of factor and the penalty parameter \(\lambda\) by cross-validation.

```
source("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/lib/cross_validation_als.R")
f_list <- seq(10, 20, 10)
l_list <- seq(-1, 0, 1)
f_l <- expand.grid(f_list, l_list)
```

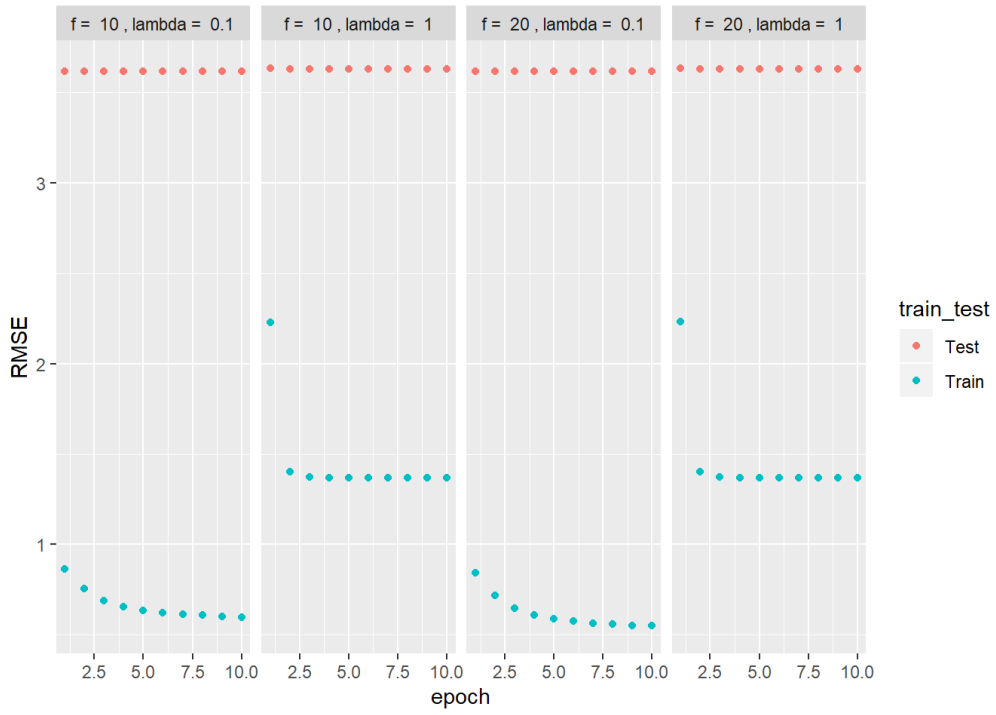
```
result_summary <- array(NA, dim = c(nrow(f_l), 10, 4))
run_time <- system.time(for(i in 1:nrow(f_l)){
    par <- paste("f = ", f_l[i,1], ", lambda = ", 10^f_l[i,2])
    cat(par, "\n")
    current_result <- cv.function(data, K = 5, f = f_l[i,1], lambda = 10^f_l[i,2])
    result_summary[,,i] <- matrix(unlist(current_result), ncol = 10, byrow = T)
    print(result_summary)

})

save(result_summary, file = "C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/rmsea3new.Rdata")
```

It takes a while to run this chunck so we have saved the output and can be loaded directly.

```
load("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/rmsea3new.Rdata")
rmse <- data.frame(rbind(t(result_summary[1,,]), t(result_summary[2,,])), train_test = rep(c("Train", "Test"), each = 4), par = rep(paste("f = ", f_l[,1], ", lambda = ", 10^f_l[,2]), times = 2)) %>% gather("epoch", "RMSE", -train_test, -par)
rmse$epoch <- as.numeric(gsub("X", "", rmse$epoch))
rmse %>% ggplot(aes(x = epoch, y = RMSE, col = train_test)) + geom_point() + facet_grid(~par)
```



Step 3 Postprocessing

After matrix factorization, postporcessing will be performed to improve accuracy. The referenced papers are:

P1:Global bias correction Section 4.1

P2:Postprocessing SVD with KNN Section 3.5

P3:Postprocessing SVD with kernel ridge regression Section 3.6

P4:Linearly combination of predictors Section 4.1

```
result <- ALS1(f = 10, lambda =0.1,
                    max.iter = 5, data = data, train = data_train, test = data_test)

save(result, file = "C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/mat_fac_a3_new.RData")
```

Knn PostProcessing

```
load("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/mat_fac_a3_new.RData")

library(lsa)
```

```
## Loading required package: SnowballC
```

```
library(plyr)
```

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

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

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

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
ratingmean<-function(data){
  mean(data$rating)
}

rating<-ddply(data,.(movieId),ratingmean)
rating$index<-c(1:nrow(rating))


l_distance<-cosine(result$Movie)
diag(l_distance)<-0
cos_id<-rep(1:I,ncol=I)
knn_r<-rep(1:I,ncol=I)
for (i in 1:I){
    cos_id[i]<-which.max(rank(l_distance[,i]))
    knn_r[i]<-rating[rating$index==cos_id[i],]$V1
  }
```

Linear Regression for Training dataset

```
data_train$movieindex<-dense_rank(data_train$movieId)

r<-rep(0,nrow(data_train))
for (i in 1:nrow(data_train)){
  rowindex<-data_train$userId[i]
  columindex<-data_train$movieindex[i]
  qi <- as.matrix(result$User[,rowindex])
  qj <- as.matrix(result$Movie[,columindex])
  r[i]<-t(qi)%*%qj
  }
w<-knn_r[data_train$movieindex]
```

```
data_train_linear<-as.data.frame(cbind(data_train$rating,r,w))
fit<-lm(V1~r+w,data=data_train_linear)
exp_rating<-predict(fit,data_train_linear)
(rmse_adj <- sqrt(mean((data_train_linear$V1 - exp_rating)^2)))
```

```
## [1] 0.5893299
```

```
cat("The RMSE train of the adjusted model is", rmse_adj)
```

```
## The RMSE train of the adjusted model is 0.5893299
```

prediction

```
data_test$movieindex<-dense_rank(data_test$movieId)

r1<-rep(0,nrow(data_test))
for (i in 1:nrow(data_test)){
  rowindex<-data_test$userId[i]
  columindex<-data_test$movieindex[i]
  qi <- as.matrix(result$User[,rowindex])
  qj <- as.matrix(result$Movie[,columindex])
  r1[i]<-t(qi)%*%qj
}

w1<-knn_r[data_test$movieindex]
```

```
data_test_linear<-as.data.frame(cbind(data_test$rating,r1,w1))
exp_rating_test<-predict(fit,data_test_linear)
rmse_adj_test <- sqrt(mean((data_test_linear$V1 - exp_rating_test)^2))
cat("The RMSE test of the adjusted model is", rmse_adj_test)
```

```
## The RMSE test of the adjusted model is 1.344724
```

Step 4 Evaluation

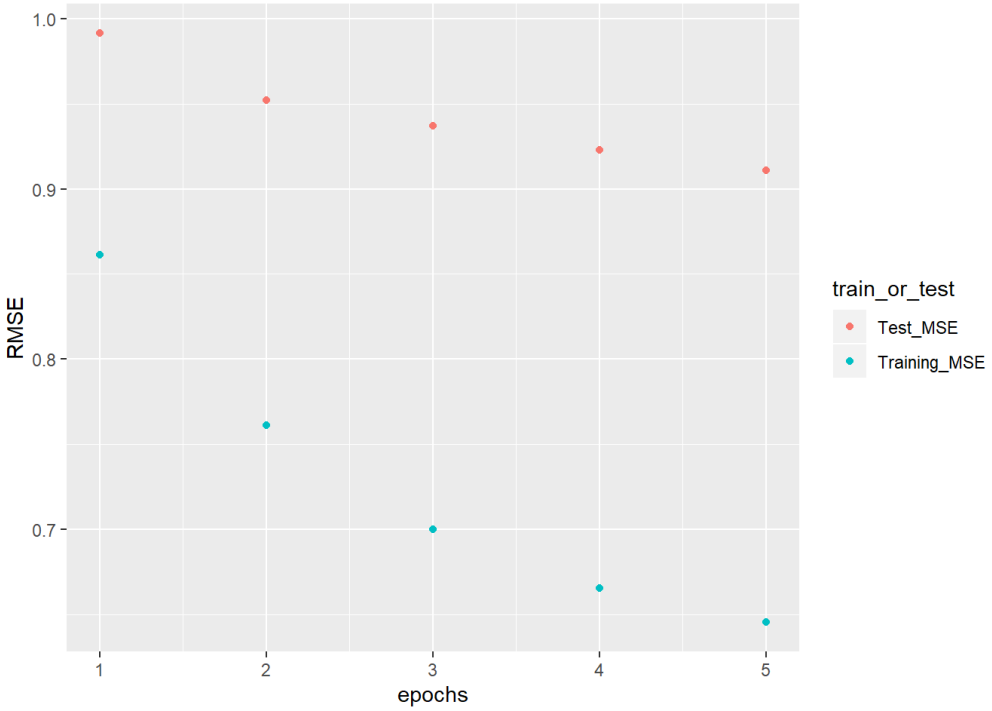
You should visualize training and testing RMSE by different dimension of factors and epochs (One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE).

```
library(ggplot2)

RMSE <- data.frame(epochs = seq(1, 5, 1), Training_MSE = result$train_RMSE, Test_MSE = result$test_RMSE) %>%
gather(key = train_or_test, value = RMSE, -epochs)

RMSE %>% ggplot(aes(x = epochs, y = RMSE,col = train_or_test)) + geom_point() + scale_x_discrete(limits = se
q(1, 5, 1)) + xlim(c(1, 5))
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```


1.0
0.9
0.8
0.7
RMSE
1
2
3
4
5
epochs
train_or_test
Test_MSE
Training_MSE

Model 2:Alternating Least Squares + Penalty of magnitudes + Bias and Intercepts + KNN

Equations recap

Alternating Least Squares

$$min_{q^\star p^\star} \sum_{(u,i)\in K} (r_u i - q_i^T p_u)^2 + \lambda(\sum_i n_{q_i}\|q_i\|^2 + \sum_i n_{p_u}\|p_u\|^2)$$

ALS technique rotate between fixing the q_i's and fixing the p_u's. When all pu's are fixed, system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each step decreases object function until convergence.

f: dimension of latent factors

q_i: factors associated with item i , measures the extent to which items possesses those factors

p_u: factors associated with user u, measures the extent of interest that user has in an item are high on corresponding factors.

Regularizations

1. Penalty of magnitudes:

$$\sum_{(u,i)\in K} \lambda(\|q_i\|^2 + \|p_u\|^2)$$

The constant lambda controls the extent of regularization and we determined it through cross validation.

2.Bias and Intercepts:

$$b_{ui} = \mu + b_i + b_u$$
$$\hat{r}_{ui} = b_{ui} + q_i^T p_u$$

$$min_{q^\star,p^\star,n^\star} \sum_{(u,i)\in K} (r_u i - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Because some users tend to giver higher or lower ratings than others, we have to add user or item bias in our object function.

Post-processing

KNN:

$$s(q_i, q_j) = \frac{q_i^T, q_j}{\|q_i\|\|q_j\|}$$

We first defined similarity between movies by the above formula. Then we used this similarity S to apply KNN prediction.

Codes and Details

Project 4 ALS R1R2

HANG HU hh2718

11/20/2019

Step 1 Load Data and Train-test Split

```
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
```

```
library(tidyr)
library(ggplot2)
data <- read.csv("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/data/ml-latest-small/ratings.csv")
set.seed(0)
## shuffle the row of the entire dataset
data <- data[sample(nrow(data)),]
## get a small dataset that contains all users and all movies
unique.user<-duplicated(data[,1])
unique.movie<-duplicated(data[,2])
index<-unique.user & unique.movie
all.user.movie <- data[!index,]

## split training and test on the rest
rest <- data[index,]
test_idx <- sample(rownames(rest), round(nrow(data)/5, 0))
train_idx <- setdiff(rownames(rest), test_idx)

## combine the training with the previous dataset, which has all users and all movies
data_train <- rbind(all.user.movie, data[train_idx,])
data_test <- data[test_idx,]

## sort the training and testing data by userId then by movieId,
## so when we update p and q, it is less likely to make mistakes
data_train <- arrange(data_train, userId, movieId)
data_test <- arrange(data_test, userId, movieId)
```

###Step 2 Matrix Factorization #### Step 2.1 Algorithm and Regularization Here I perform stochastic gradien descent to do matrix factorization. Your algorithm should consider case that there are new users and movies adding to the dataset you used to train. In other words, the dimension your matrix R, q, p is dynamic.

- For algorithms, the referenced paper are:

A1. Stochastic Gradient Descent Section: Learning Algorithms-Stochastic Gradient Descent

A2. Gradient Descent with Probabilistic Assumptions Section 2

A3. Alternating Least Squares Section 3.1

- For regularizations, the referenced paper are:

R1. Penalty of Magnitudes Section: a Basic Matrix Factorization Model

R2. Bias and Intercepts Section: Adding Biases

R3. Temporal Dynamics Section 4

```
U <- length(unique(data$userId))
I <- length(unique(data$movieId))
source("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/lib/ALS.R1+R2.R")
```

Step 2.2 Parameter Tuning

Here you should tune parameters, such as the dimension of factor and the penalty parameter \(\lambda\) by cross-validation.

```
source("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/lib/cross_validation_R1R2.R")
f_list <- seq(10, 20, 10)
l_list <- seq(1, 2, 1)
f_l <- expand.grid(f_list, l_list)
```

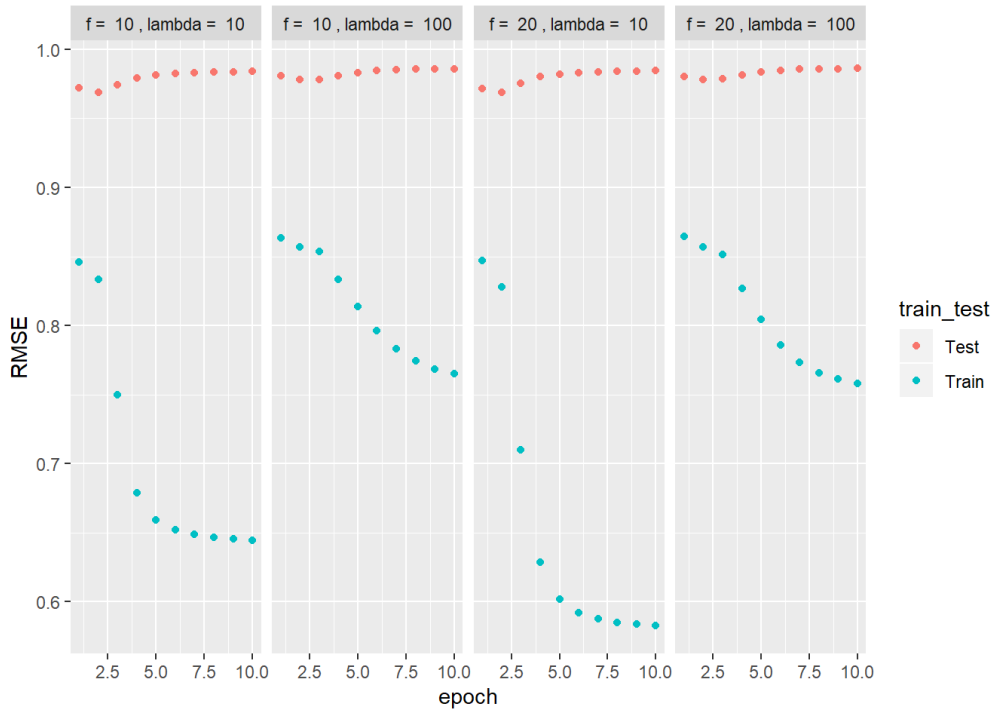
It takes time for running again and result has been saved.

```
result_summary <- array(NA, dim = c(nrow(f_l), 10, 4))
run_time <- system.time(for(i in 1:nrow(f_l)){
    par <- paste("f = ", f_l[i,1], ", lambda = ", 10*f_l[i,2])
    cat(par, "\n")
    current_result <- cv.function(data, K = 5, f = f_l[i,1], lambda = 10*f_l[i,2])
    result_summary[,,i] <- matrix(unlist(current_result), ncol = 10, byrow = T)
    print(result_summary)

})

save(result_summary, file = "C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/rmser1r2new.Rdata")
```

```
load("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/rmser1r2new.Rdata")
rmse <- data.frame(rbind(t(result_summary[1,,]), t(result_summary[2,,])), train_test = rep(c("Train", "Test"), each = 4), par = rep(paste("f = ", f_l[,1], ", lambda = ", 10^f_l[,2]), times = 2)) %>% gather("epoch", "RMSE", -train_test, -par)
rmse$epoch <- as.numeric(gsub("X", "", rmse$epoch))
rmse %>% ggplot(aes(x = epoch, y = RMSE, col = train_test)) + geom_point() + facet_grid(~par)
```



Step 3 Postprocessing

After matrix factorization, postporcessing will be performed to improve accuracy. The referenced papers are:

P1:Global bias correction Section 4.1

P2:Postprocessing SVD with KNN Section 3.5

P3:Postprocessing SVD with kernel ridge regression Section 3.6

P4:Linearly combination of predictors Section 4.1

```
result <- ALS.R1R2(f = 20, lambda =10,
                   max.iter = 4, data = data, train = data_train, test = data_test)

save(result, file = "C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/mat_fac_r1r2new.RData")
```

Knn PostProcessing

```
load("C:/Users/huhan/OneDrive/Desktop/2019 Fall/GR 5243/fall2019-project4-sec2-grp2-master/fall2019-project4-sec2-grp2-master/output/mat_fac_r1r2new.RData")
library(lsa)
```

```
## Loading required package: SnowballC
```

```
library(plyr)
```

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

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

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

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
ratingmean<-function(data){
  mean(data$rating)
}

rating<-ddply(data,.(movieId),ratingmean)
rating$index<-c(1:nrow(rating))


l_distance<-cosine(result$q)
diag(l_distance)<-0
cos_id<-rep(1:I,ncol=I)
knn_r<-rep(1:I,ncol=I)
for (i in 1:I){
    cos_id[i]<-which.max(rank(l_distance[,i]))
    knn_r[i]<-rating[rating$index==cos_id[i],]$V1
  }
```

Linear Regression for Training dataset

```
data_train$movieindex<-dense_rank(data_train$movieId)

r<-rep(0,nrow(data_train))
for (i in 1:nrow(data_train)){
  rowindex<-data_train$userId[i]
  columindex<-data_train$movieindex[i]
  qi <- as.matrix(result$p[,rowindex])
  qj <- as.matrix(result$q[,columindex])
  r[i]<-t(qi)%*%qj
  }
```

```
v<-rep(0,nrow(data_train))
for (i in 1:nrow(data_train)){
  rowindex<-data_train$userId[i]
  columindex<-data_train$movieindex[i]
  qi <- as.matrix(result$bu[rowindex])
  qj <- as.matrix(result$bi[columindex])
  v[i]<-qi+qj+result$mu
}

w<-knn_r[data_train$movieindex]
```

```
data_train_linear<-as.data.frame(cbind(data_train$rating,r,v,w))
fit<-lm(V1~r+v+w,data=data_train_linear)
exp_rating<-predict(fit,data_train_linear)
(rmse_adj <- sqrt(mean((data_train_linear$V1 - exp_rating)^2)))
```

```
## [1] 0.5925521
```

```
cat("The RMSE train of the adjusted model is", rmse_adj)
```

```
## The RMSE train of the adjusted model is 0.5925521
```

prediction

```
data_test$movieindex<-dense_rank(data_test$movieId)

r1<-rep(0,nrow(data_test))
for (i in 1:nrow(data_test)){
  rowindex<-data_test$userId[i]
  columindex<-data_test$movieindex[i]
  qi <- as.matrix(result$p[,rowindex])
  qj <- as.matrix(result$q[,columindex])
  r1[i]<-t(qi)%*%qj
  }
```

```
v1<-rep(0,nrow(data_test))
for (i in 1:nrow(data_test)){
  rowindex<-data_test$userId[i]
  columindex<-data_test$movieindex[i]
  qi <- as.matrix(result$bu[rowindex])
  qj <- as.matrix(result$bi[columindex])
  v1[i]<-qi+qj+result$mu
}

w1<-knn_r[data_test$movieindex]
```

```
data_test_linear<-as.data.frame(cbind(data_test$rating,r1,v1,w1))
exp_rating_test<-predict(fit,data_test_linear)
rmse_adj_test <- sqrt(mean((data_test_linear$V1 - exp_rating_test)^2))
cat("The RMSE test of the adjusted model is", rmse_adj_test)
```

```
## The RMSE test of the adjusted model is 1.342585
```

Step 4 Evaluation

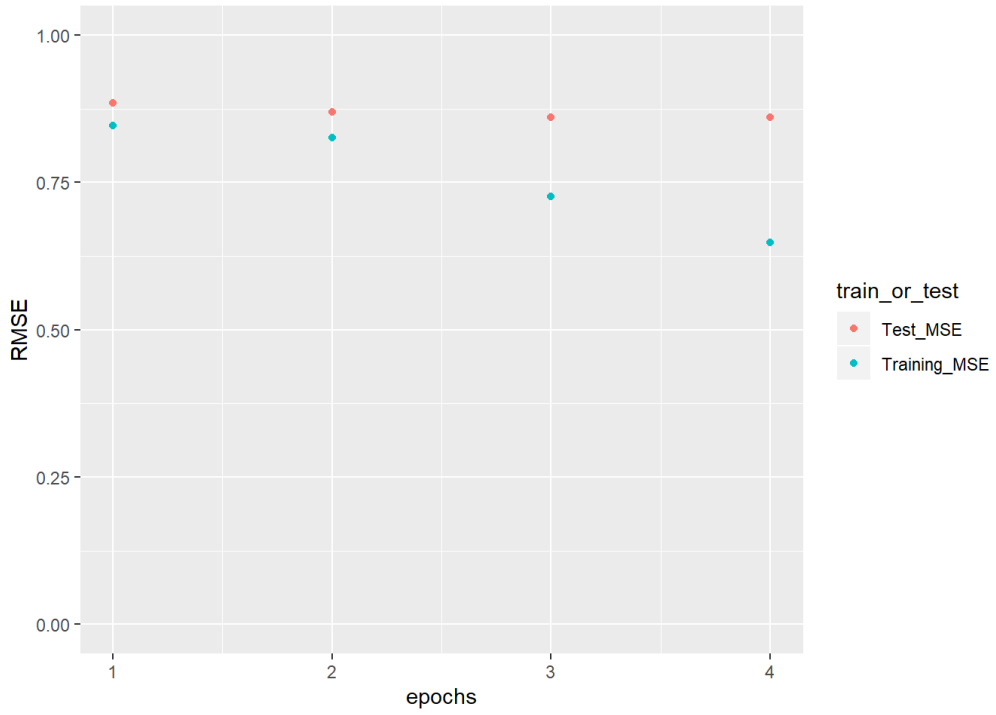
You should visualize training and testing RMSE by different dimension of factors and epochs (One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE).

```
library(ggplot2)

RMSE <- data.frame(epochs = seq(1, 4, 1), Training_MSE = result$train_RMSE, Test_MSE = result$test_RMSE) %>%
gather(key = train_or_test, value = RMSE, -epochs)

RMSE %>% ggplot(aes(x = epochs, y = RMSE,col = train_or_test)) + geom_point() + scale_x_discrete(limits = se
q(1, 5, 1)) + xlim(c(1, 4))+ylim(c(0,1))
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



Model 3:Alternating Least Squares + Temporal Dynamics + KNN

Equations recap

Alternating Least Squares

$$min_{q^\star p^\star} \sum_{(u,i)\in K} (r_u i - q_i^T p_u)^2 + \lambda(\sum_i n_{q_i}\|q_i\|^2 + \sum_i n_{p_u}\|p_u\|^2)$$

ALS technique rotate between fixing the q_i's and fixing the p_u's. When all pu's are fixed, system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each step decreases object function until convergence.

f: dimension of latent factors

q_i: factors associated with item i , measures the extent to which items possesses those factors

p_u: factors associated with user u, measures the extent of interest that user has in an item are high on corresponding factors.

Regularizations

Temporal Dynamics:

$$\hat{r}_u i = \mu + b_u + b_i + b_{i.Bin(t)} + q_i^T p_u$$

Since users' preferences and products' popularity are changing over time. We have to consider time factor so add a new variable time to our previous function.For each bin, we used one year of the day and we get 23 bins in total.

Post-processing

KNN:

$$s(q_i, q_j) = \frac{q_i^T, q_j}{\|q_i\|\|q_j\|}$$

We first defined similarity between movies by the above formula. Then we used this similarity S to apply KNN prediction.

Codes and Details

A3 + R3

Code

Chang Xu

Load data, split dataset

Hide

```
data <- read.csv("../data/ml-latest-small/ratings.csv")
```

Hide

```
data$timestamp <- anydate(data$timestamp)
data <- arrange(data, timestamp)
```

Hide

```
bins <- as.numeric(max(data$timestamp) - min(data$timestamp))
cut <- seq(0, 1, by=1/30)
quan <- quantile(c(1:bins), cut)
Bin <- rep(NA, nrow(data)-1)
for (i in 1:nrow(data)) {
    dev <- as.numeric(data$timestamp[i] - data$timestamp[1]) + 1
    for (j in 1:30) {
        if (quan[j] <= dev & dev <= quan[j+1]) Bin[i] = j
    }
}
data$Bin <- c(Bin, 30)
```

Hide

```
set.seed(0)
## shuffle the row of the entire dataset
data <- data[sample(nrow(data)),]
## get a small dataset that contains all users and all movies
unique.user<-duplicated(data[,1])
unique.movie<-duplicated(data[,2])
index<-unique.user & unique.movie
all.user.movie <- data[!index,]

## split training and test on the rest
rest <- data[index,]
test_idx <- sample(rownames(rest), round(nrow(data)/5, 0))
train_idx <- setdiff(rownames(rest), test_idx)

## combine the training with the previous dataset, which has all users and all movies
data_train <- rbind(all.user.movie, data[train_idx,])
data_test <- data[test_idx,]

## sort the training and testing data by userId then by movieId,
## so when we update p and q, it is less likely to make mistakes
data_train <- arrange(data_train, userId, movieId)
data_test <- arrange(data_test, userId, movieId)
```

Hide

```
U <- length(unique(data$userId))
I <- length(unique(data$movieId))
```

define RMSE

Hide

```
RMSE <- function(rating, est_rating) {
  error <- rep(NA, nrow(rating))
  for (i in 1:nrow(rating)) {
    a <- as.character(rating[i,1])
    b <-as.character(rating[i,2])
    error[i] <- (rating[i,3]-est_rating[a, b])^2
  }
  return(sqrt(mean(error)))
}
```

A3 + R3

Hide

Hide

```
# Alternating least squares
# a function returns a list containing factorized matrices p and q, training and test
ing RMSEs.

# ALS.R3 <- function(f = 4, lambda = 0.1, max.iter = 3,
#                   data = data, train = data_train, test = data_test){
f <- 4
lambda <- 0.1
max.iter <- 3
data <- data
train <- data_train
test <- data_test
  # Step 1: Initialize Movie matrix (q), User matrix (p), Movie bias(bi) and User bia
s(bu)

  # Step 1: Initialize Movie matrix (q), User matrix (p), Movie bias(bi) and User bia
s(bu)
  p <- matrix(runif(f*U, -10, 10), ncol = U)
  colnames(p) <- levels(as.factor(data$userId))

  q <- matrix(runif(f*I, -10, 10), ncol = I)
  colnames(q) <- levels(as.factor(data$movieId))

  bu <- matrix(rep(0, U), ncol=U)
  colnames(bu) <- levels(as.factor(data$userId))

  bi <- matrix(rep(0, I), ncol = I)
  colnames(bi) <- levels(as.factor(data$movieId))

  R <- matrix(rep(0, U*I), ncol = I)
  colnames(R) <- levels(as.factor(data$movieId))

  bit <- matrix(rep(NA, I*30), ncol = I)
  colnames(bi) <- levels(as.factor(data$movieId))

  #mean of all the ratings in train data set.
  mu <- mean(train$rating)

  # sort movie id to make sure that they are from small to large
  movie.id <- sort(unique(data$movieId))

  # sort the data by userid then by movie id
  train <- arrange(train, userId, movieId)



  # set vacant cells to record train and test rmse
```

```
  train_RMSE <- c()
  test_RMSE <- c()

 for (l in 1:max.iter) {
    # Step 2: Fix q, solve p
    # we need new factors to add bu, bi into calculation
    q_idb <- rbind(rep(1,I), q)
    colnames(q_idb) <- levels(as.factor(data$movieId))
    p_idb <- rbind(bu, p)

    for (u in 1:U) {
      # find all the moives rated by user u
      i_ratedby_u <- as.character(train[train$userId==u,]$movieId)
      x<-train[train$userId==u,]$rating
      R_m_u <- matrix(x,ncol=length(x),nrow = 1)
      # update p.tilde
      p_idb[,u] <- solve(q_idb[,i_ratedby_u] %*% t(q_idb[,i_ratedby_u]) + lambda * di
ag(f+1)) %*%
        q_idb[,i_ratedby_u] %*% t(R_m_u - mu -bi[,i_ratedby_u])
    }

    # update bu and p
    bu[1,] <- p_idb[1, ]
    p <- p_idb[-1, ]

    # Step 3: Fix p, solve q
    # we need new factors to add bu, bi into calculation
    p.tilde <- rbind(rep(1,U), p)
    colnames(p.tilde) <- levels(as.factor(data$userId))
    q.tilde <- rbind(bi, q)

    for (i in 1:I) {
      # find all the users who rate movie i
      u.rated.i <- as.character(train[train$movieId==movie.id[i],]$userId)
      q.tilde[,i] <- solve(p.tilde[,u.rated.i] %*% t(p.tilde[,u.rated.i]) + lambda* d
iag(f+1)) %*%
        p.tilde[,u.rated.i] %*% (train[train$movieId==movie.id[i],]$rating - mu - bu[
,u.rated.i])

    }

    # update bi and q
    bi[1,] <- q.tilde[1,]
    q <- q.tilde[-1,]

    # update bit
    for (t in 1:30) {
```

```
  sub <- filter(train, Bin == t)
  for (i in 1:I) {
    ssub <- filter(sub, movieId == i)
    r <- sum(ssub$rating)
    n <- length(unique(ssub$userId))+lambda
    bit[t, i] <- r/n
  }
}

# Rating Matrix
mat <- matrix(rep(NA, U*I), ncol = I)
for (u in 1:U) {
  for (i in 1:I) {
    mat[u, i] <- as.numeric(t(p[,u]) %*% q[,i])
  }
}

bu_ui <- matrix(rep(NA, U*I), ncol = I)
for (i in 1:I) {
  bu_ui[,i] <- t(bu)
}

bi_ui <- matrix(rep(NA, U*I), ncol = I)
for (u in 1:U) {
  bi_ui[u, ] <- bi
}

bit_ui <- matrix(rep(0, U*I), ncol = I)
for (u in 1:U) {
  for (i in 1:I) {
      sub <- filter(train, userId == u, movieId == i)
      if (dim(sub)[1] > 0) {
        t <- as.numeric(sub$Bin)
        bit_ui[u, i] <- bit[t, i]
      }
    }
  }

mu_ui <- matrix(rep(mu, U*I), ncol = I)

R <- mu_ui + bu_ui + bi_ui + mat + bit_ui

# Summerize
cat("iter:", l, "\t")
est_rating <- as.matrix(R)
colnames(est_rating) <- levels(as.factor(data$movieId))
rownames(est_rating) <- levels(as.factor(data$userId))
```

```
    train_RMSE_cur <- RMSE(train, est_rating)
    cat("training RMSE:", train_RMSE_cur, "\t")
    train_RMSE <- c(train_RMSE, train_RMSE_cur)

    test_RMSE_cur <- RMSE(test, est_rating)
    cat("test RMSE:",test_RMSE_cur, "\n")
    test_RMSE <- c(test_RMSE, test_RMSE_cur)
 }
```

Hide

```
save(ALS.R3, file = "../output/rmsea3r3.RData")
```

cross validation

Hide

```
source("../lib/cross_validation.R")
f_list <- seq(10, 20, 10)
l_list <- seq(-2, -1, 1)
f_l <- expand.grid(f_list, l_list)
```

Hide

```
result_summary <- array(NA, dim = c(nrow(f_l), 10, 4))
run_time <- system.time(for(i in 1:nrow(f_l)){
    par <- paste("f = ", f_l[i,1], ", lambda = ", 10^f_l[i,2])
    cat(par, "\n")
    current_result <- cv.function(data, K = 5, f = f_l[i,1], lambda = 10^f_l[i,2])
    result_summary[,,i] <- matrix(unlist(current_result), ncol = 10, byrow = T)
    print(result_summary)

})

save(result_summary, file = "../output/rmse.Rdata")
```

Hide

```
load("../output/rmse.Rdata")
rmse <- data.frame(rbind(t(result_summary[1,,]), t(result_summary[2,,])), train_test
= rep(c("Train", "Test"), each = 4), par = rep(paste("f = ", f_l[,1], ", lambda = ",
10^f_l[,2]), times = 2)) %>% gather("epoch", "RMSE", -train_test, -par)
rmse$epoch <- as.numeric(gsub("X", "", rmse$epoch))
rmse %>% ggplot(aes(x = epoch, y = RMSE, col = train_test)) + geom_point() + facet_gr
id(~par)
```

KNN

Hide

```
result <- gradesc(f = 10, lambda = 0.1,lrate = 0.01, max.iter = 100, stopping.deriv =
0.01,
                  data = data, train = data_train, test = data_test)

save(result, file = "../output/mat_fac.RData")
```

Hide

```
load(file = "../output/mat_fac.RData")

pred_rating <- t(result$q) %*% result$p
#define a function to extract the corresponding predictedrating for the test set.
extract_pred_rating <- function(test_set, pred){
  pred_rating <- pred[as.character(test_set[2]), as.character(test_set[1])]
  return(pred_rating)
}
#extract predicted rating
pred_test_rating <- apply(data_test, 1, extract_pred_rating, pred_rating)

#mean(P)
pred_mean <- mean(pred_test_rating)
#mean(test)
mean_test_rating <- mean(data_test$rating)

#mean(test) - mean(P)
mean_diff <- mean_test_rating - pred_mean

data_test$pred <- pred_test_rating
data_test$pred_adj <- pred_test_rating + mean_diff

boxplot(data_test$pred_adj ~ data_test$rating)
#calculate RMSE
rmse_adj <- sqrt(mean((data_test$rating - data_test$pred_adj)^2))
cat("The RMSE of the adjusted model is", rmse_adj)
```

Conclusions

Methods	f	λ	Max Iteration	Train RMSE	Test RMSE
A3+P2	10	0.1	5	0.5893	1.3447
A3+R1R2+P2	20	10	4	0.5926	1.3426
A3+R3+P2	10	5	5	0.82	NA