Project 4 Report

Group 12

In this project, we applied the following methods for collaborative filtering:

- factorization algorithm: Stochastic Gradient Descent
- regularization: Temporal Dynamics
- postpocessing: KNN & Kernel Ridge Regression

Model Setup: Let

$$\hat{r_{ui}} = \mu + b_i(t) + b_u + q_i^T p_u$$

where

$$b_i(t) = b_i + b_{i,Bin(t)}$$

and μ is the average of all ratings.

The objective function is then:

$$Minimze \quad \sum (r_{ui} - \mu - b_i - b_{i,Bin(t)} - b_u - q_i^T p_u) + \lambda(||q_i^2|| + ||p_u^2|| + b_i^2 + b_{i,Bin(t)}^2 + b_u^2)$$

Our goal is to minimize the objective function using stochastic gradient descent, apply the estimated parameters to two different postporcessing process, and compare their results.

Step 1 Load Data and Train-test Split

```
library(dplyr)
library(tidyr)
library(ggplot2)
library(anytime)
library(ggplot2)
library(lubridate)
run_all <- F
data <- read.csv("../data/ml-latest-small/ratings.csv")
set.seed(0)

# Function for dividing time into bins, and add bins to the original data

bins <- function(data = data){
  data$date <- anytime(data$timestamp) %>% format("%Y/%m/%d")
  df <- data %>% mutate(year = year(date), bin_label = year-1995) %>%
    data.frame() %>%
    select(userId, movieId, rating, timestamp, bin_label, date, year)
  min_day <- df %>% summarise(as.Date(min(date))) %>% as.numeric()
  df <- df %>% mutate(days_diff=as.numeric(as.Date(date)-min_day)+1) %>%
    group_by(userId) %>% mutate(user_mean_date=mean(days_diff))
  return(df)
}
```

```
data_bins <- bins(data=data)
```

Step 2 Matrix Factorization

Step 2.1 Algorithm and Regularization

- Algorithm: Stochastic Gradient Descent (A1)
- Regularizations: Temporal Dynamics (R3)

First, we take a subset of 5000 samples in the original data to do the cross validation because it takes too long to do the cross validation on the whole data set.

```
index <- sample(1:100000, 5000)
test_idx <- sample(1:5000, 1000)
train_idx <- setdiff(1:5000, test_idx)
data_train <- data_bins[train_idx,]
data_test <- data_bins[test_idx,]
U <- length(unique(data_bins[index,]$userId)) #610
I <- length(unique(data_bins[index,]$movieId)) #9724
source("../lib/Matrix_Factorization.R")
```

Step 2.2 Parameter Tuning

Here we use cross validation to tune the 2 hyperparameters: f and lambda, and select the optimal combination by choosing the one with lowest test RMSE.

```
# generate a data frame contains all the (f, lambda) pairs to do the cross validation

source("../lib/cross_validation.R")
f_list <- c(10, 20, 50)
l_list <- seq(-3,-1,1)
f_l <- expand.grid(f_list, l_list)
```

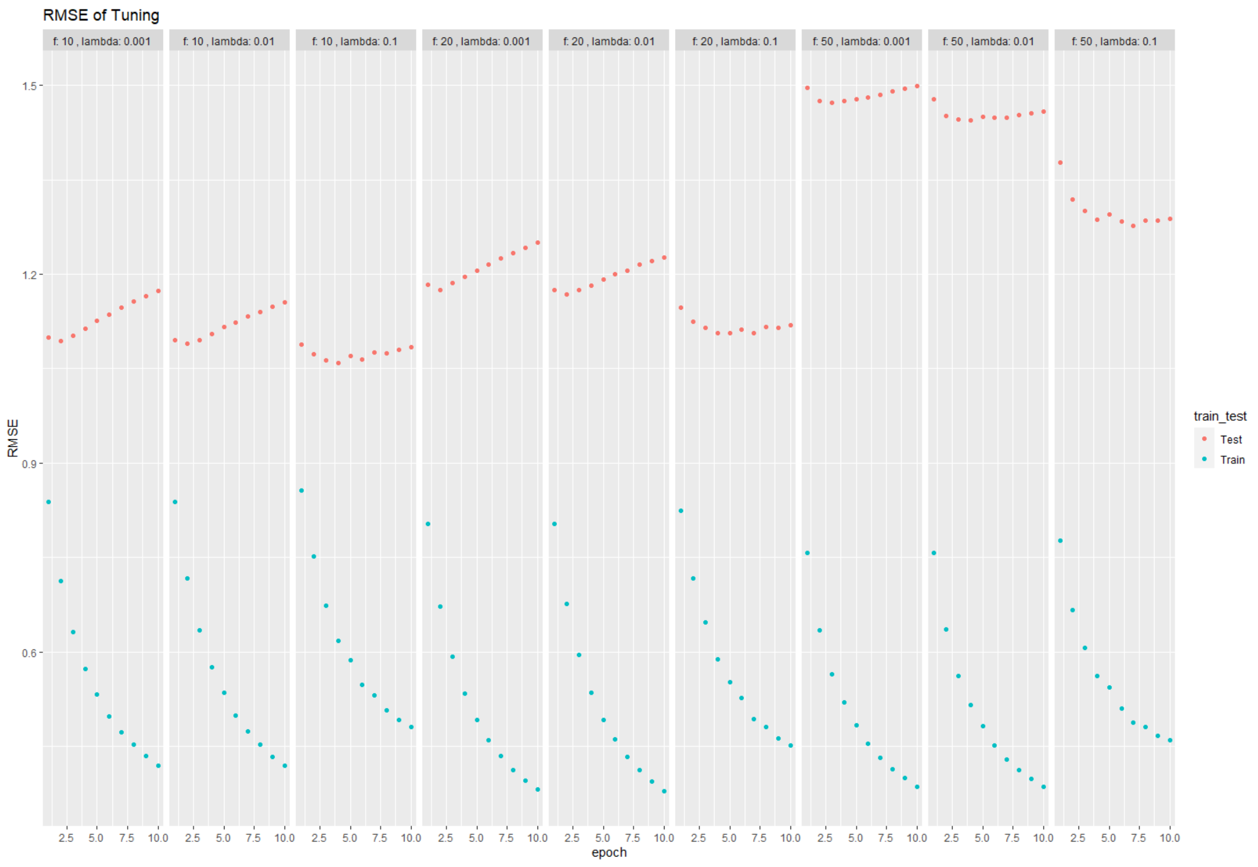
```
#time1 <- Sys.time()
#6.59713 hours
if(run_all){
  result_summary <- array(NA, dim = c(4, 10, nrow(f_l)))
run_time <- system.time(for(i in 1:nrow(f_l)){
    par <- paste("f = ", f_l[i,1], ", lambda = ", 10^f_l[i,2])
    cat(par, "\n")
    current_result <- cv.function_r3(data_bins[index,], K = 3, 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_r3)
})
#time2 <- Sys.time()
#time2-time1
save(result_summary, file = "../output/rmse_r3.Rdata")
}
```

```
# plot and compare the result of cross validation

if(run_all){
load("../output/rmse_r3.Rdata")
rmse_r3 <- data.frame(rbind(t(result_summary[1,,]), t(result_summary[2,,])),
                   train_test = rep(c("Train", "Test"), each = 9),
                   par = rep(paste("f:", f_l[,1], ", lambda:", 10^f_l[,2]), times = 2)) %>%
  gather("epoch", "RMSE", -train_test, -par)

rmse_r3$epoch <- as.numeric(gsub("X", "", rmse_r3$epoch))

ggplot(rmse_r3, aes(x = epoch, y = RMSE, col = train_test)) + geom_point() +
  facet_grid(~par)+ggtitle("RMSE of Tuning")}
```



```
# because we only cross validate on the subset of the original data
# here we split the data for training the model

test_idx <- sample(1:nrow(data_bins), round(nrow(data_bins)/5, 0))
train_idx <- setdiff(1:nrow(data), test_idx)
data_train <- data_bins[train_idx,]
data_test <- data_bins[test_idx,]
U <- length(unique(data_bins$userId)) #610
I <- length(unique(data_bins$movieId)) #9724
```

After finding the best parameters combination (f = 10, lambda = 0.1), we run Stochastic Gradient Descent to estimate all the parameters in the objective function which includes b_i, $b_{i,Bin(t)}$, b_u, p_u and q_i.

```
if (run_all){
    result <- gradesc.r3(f = 10, lambda = 0.1,lrate = 0.01, max.iter = 100,
                         stopping.deriv = 0.01,
                         data = data_bins, train = data_train, test = data_test)
  save(result, file = "../output/mat_fac_r3.RData")
  write.csv(result$p, file = "../output/p.csv")
  write.csv(result$q, file = "../output/q.csv")
  write.csv(result$b_user, file = "../output/b_user.csv")
  write.csv(result$b_movie, file = "../output/b_movie.csv")
  write.csv(result$b_bin, file = "../output/b_bin.csv")
  write.csv(result$mu, file = "../output/mu.csv")
}
```

Step 3 Postprocessing

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

P2:Postprocessing SVD with KNN Section 3.5

P3:Postprocessing SVD with kernel ridge regression Section 3.6

Postprocessing: SVD with KNN

The first Postprocessing method we use is KNN. Notice that we only need matrix q in the previous sections to do the postprocessing.

Here is the definition of KNN postprocessing function.

```
vec <- function(x) {return(sqrt(sum(x^2)))}

pred_knn <- function(data_train, data_test, q)
{
  norm_q <- apply(q,2,vec)
  sim <- t(t((t(q) %*% q)/ norm_q) / norm_q)
  colnames(sim) <- colnames(q)
  rownames(sim) <- colnames(q)
  pred_test <- rep(0,nrow(data_test))

  for (i in 1:nrow(data_test)){
    user_id <- data_test$userId[i]
    movie_id <- data_test$movieId[i]
    train <- data_train[data_train$userId == user_id & data_train$movieId != movie_id,]
    movie_train <- train$movieId
    sim_vec <- sim[rownames(sim) == movie_id, colnames(sim) %in% movie_train]
    movie <- names(sim_vec)[which.max(sim_vec)]
    pred_test[i] <- train[train$movieId == movie,][3]
  }
  pred_test <- as.matrix(unlist(pred_test))
  rmse_test <- sqrt(mean((data_test$rating-pred_test)^2))
  return(list(pred_test = pred_test, rmse_test = rmse_test))
}
```

```
# load the result from stochastic gradient descent algorithm and do the KNN postprocessing

load(file = "../output/mat_fac_r3.RData")
if (run_all){
  q <- result$q
  p2_result_test <- pred_knn(data_train, data_test, q)
  p2_result_train <- pred_knn(data_train, data_train, q)
  train_rmse_p2 <- p2_result_train['rmse_test']
  test_rmse_p2 <- p2_result_test['rmse_test']
  save(train_rmse_p2, file = "../output/train_rmse_p2.Rdata")
  save(test_rmse_p2, file = "../output/test_rmse_p2.Rdata")
}
```

```
load(file = "../output/train_rmse_p2.RData")
load(file = "../output/test_rmse_p2.RData")
print(paste("The RMSE of the train set with Postprocessing (SVD with KNN) is", 1.211423))
```

```
## [1] "The RMSE of the train set with Postprocessing (SVD with KNN) is 1.211423"
```

```
print(paste("The RMSE of the test set with Postprocessing (SVD with KNN) is", 1.201936))
```

```
## [1] "The RMSE of the test set with Postprocessing (SVD with KNN) is 1.201936"
```

Postprocessing: SVD with kernel ridge regression

The second Postprocessing method we use is kernel ridge regression. In kernel ridge regression, we need matrix q and true ratings in the previous sections and implement kernel ridge regression between ratings (as response variable) and q (as design matrix).

```
# normalize the matrix q by column

#library(krr)
library("pracma")
norm_vec <- function(x) {return(x/Norm(x))}
norm_q <- apply(result$q, 2, norm_vec)

# separate movieId from q dataset
movieID <- colnames(q)
```

Tuning parameter for krr

We set the kernel as Gaussian as Paper2 said and use cross validation to tune in parameter lambda in kernel ridge regression. The definition of cross validation function for parameter lambda is as follows.

```
cv.krr <- function(dat_train,K, lambda){
    n <- dim(dat_train)[1]
    n.fold <- round(n/K, 0)
    set.seed(0)
    s <- sample(rep(1:K, c(rep(n.fold, K-1), n-(K-1)*n.fold)))
    test_rmse<-rep(0,K)

for (j in 1:K){
    train.data <- dat_train[s != j,]
    test.data <- dat_train[s == j,]
    tMSE <- 0
for(i in 1:610){
   userID = as.character(i)

   movie_train_index <-
     which(movieID %in% train.data$movieId[which(train.data$userId == userID)])
   movie_test_index <-
     which(movieID %in% test.data$movieId[which(test.data$userId == userID)])
   obj <- krr(t(norm_q[, movie_train_index]), train.data$rating[movie_train_index],lambda)
   xnew <- norm_q[, movie_test_index]
   ynew <- predict(obj, t(xnew))
```

```r
   ytrue <- test.data$rating[which(test.data$userId == userID)]
   tMSE <- tMSE + sum((ynew - ytrue)^ 2)
 }
   test_rmse[j]<-sqrt(tMSE/nrow(test.data))}

return(mean(test_rmse))}
```

```r
# find the best parameter that get a smaller test rmse

if(run_all){
    lambdas <- c(4.0, 5.0, 6.0)
    rmse_tune <- data.frame(lambdas = numeric(), rmse = numeric())
    #rmse_tune <- data.frame()
    colnames(rmse_tune) <- c("lambda", "rmse")

for (f in 1:length(lambdas)){
    m <- cv.krr(data_train, 5, lambdas[f])
    rmse_tune <- rbind(rmse_tune, c(lambdas[f], m))
}
colnames(rmse_tune) <- c("lambda", "rmse")

min_rmse <- rmse_tune %>% filter(rmse == min(rmse))
best_para <- min_rmse[,1]}
# best_para  lambda = 4
```

```r
# use the best lambda to train a total of 610 kernel ridge regression models. one for each user

t0 <- Sys.time()
if(run_all){
train_model <- vector(mode="list",length = 610)
for(i in 1:610){
  userID = as.character(i)

  movie_train_index <-
          which(movieID %in% data_train$movieId[which(data_train$userId == userID)])
  train_model[[i]] <- krr(t(norm_q[, movie_train_index]),
                          data_train$rating[movie_train_index],best_para)
}

t1 <- Sys.time()

training_time <- t1 - t0
training_time
}
# Time difference of 56.20646 secs
```

```r
# compute the overall rmse

if(run_all){
rmse.fn <- function(data){
  error <- 0
  for (i in 1:610){
    userID = as.character(i)
    movie_test_index <- which(movieID %in% data$movieId[which(data$userId == userID)])
    xnew <- norm_q[, movie_test_index]
    ynew <- predict(train_model[[i]], t(xnew))
    ytrue <- data$rating[which(data$userId == userID)]
    error <- error + sum((ynew - ytrue)^ 2)
  }
  return(sqrt(error / nrow(data)))
}


test_rmse <- rmse.fn(data_test)
train_rmse <- rmse.fn(data_train)}
#test_rmse 1.32
#train_rmse 1.25

print(paste("The RMSE of the train set with Postprocessing (SVD with kernel ridge regression) is",
            1,25))
```

```
## [1] "The RMSE of the train set with Postprocessing (SVD with kernel ridge regression) is 1 25"
```

```
print(paste("The RMSE of the test set with Postprocessing (SVD with kernel ridge regression) is",
            1.32))
```

```
## [1] "The RMSE of the test set with Postprocessing (SVD with kernel ridge regression) is 1.32"
```

Step 4 Evaluation

In terms of test rmse, We can see that SVD with KNN is better than SVD with Kernel Ridge Regression. More interestingly, the Stochastic Gradient Descent without postprocessing can achieve better test RMSE than both postprocessing methods. We assume that this is because when using postprocessing methods, we abandon several parameters related to temporal effects (which is an important part in the loss function).

In terms of computational efficiency, the time of cross validation takes about 6 hours and the time of stochastic gradient descent may take about 10 minutes. The time of postprocessing is always within several minutes, which is very fast.

```
# generate the table that compares the test rmse

table<- tibble(method = c('without postprocessing','KNN','Kernel Ridge Regression'),
               train_rmse = c(0.50,1.21, 1.25), test_rmse = c(1.06,1.20, 1.32))
table
```

```
## # A tibble: 3 x 3
##   method                  train_rmse test_rmse
##   <chr>                        <dbl>     <dbl>
## 1 without postprocessing        0.5       1.06
## 2 KNN                           1.21      1.2
## 3 Kernel Ridge Regression       1.25      1.32
```