Project 4

Group 4

Introduction

In this project, our group implement matrix factorization by focusing on different versions of alternating least squares algorithm for different Regularizations and Kernel Ridge Regression as Post-processing. The objective is to produce a good prediction of users' preferences for movies on on the MovieLens dataset. For this project, our team #4 is assigned with Pairing combination 12 + 14 from the Collaborative. For evaluation, we compared RMSE results for different methods. Our group used R language to product model and reports. The specific technique used for the combination are illustrated below. For more details for methods, please also see (https://github.com/TZstatsADS/ADS_Teaching/blob/master/Projects_StarterCodes/Project4-RecommenderSystem/doc/Proj4_pairings_2020_Spring.pdf).

Models

`Alternating Least Squares + Kernel Ridge Regression`

`Alternating Least Squares + Penalty of magnitudes + Temporal Dynamics + Kernel Ridge Regression`

Model 1: Alternating Least Squares + Kernel Ridge Regression

Step 1. Load Data and Train-Test Split

In this step, we loaded the data and added variable timediff for temporal dynamics, which is the difference between the user's rating time and the average rating time of this user. Then we splitted the original dataset to training set(80%) and test set(20%)

```
# Convert timestamp and set up time difference
data <- read.csv("../data/ml-latest-small/ratings.csv")
data$timestamp <- anydate(data$timestamp)
data_R <- data %>%
  group_by(userId) %>%
  mutate(timediff = (timestamp - mean(timestamp)) %>%
           as.numeric) %>%
  ungroup() %>%
  arrange(timestamp)

set.seed(1)

# train-test split (.8/.2)
train_ind <- createDataPartition(data$userId, p=.8, list=F)

data_train <- data_R[c(train_ind), ]
data_test <- data_R[-c(train_ind), ]
```

```
train <- data_train[, 1:3]
test <- data_test[, 1:3]
```

Step 2. Model Setup

1. 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

Kernel Ridge Regression:

$$\hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$$
$$\hat{y_i} = {x_i}^T \hat{\beta}$$

Equilvalent to:

$$\hat{\beta} = X^T (X X^T + \lambda I)^{-1} y$$

$$\hat{y}_i = K(x_i^T, X)(K(X, X) + \lambda I)^{-1} y$$

y: vector of movies rated by user i X: a matrix of observations - each row of X is normalized vector of features of one movie j rated by user i:

$$x_{j_2} = \frac{q_j}{\|q_j\|}$$

K:

$$K(x_i^T, X) = x_i^T x_j$$
$$K(x_i^T, X) = \exp(2(x_i^T x_j - 1))$$

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

2. Algorithm Setup

For matrix factorization, with the method in paper4, we developed our algorithm. To speed up the algorithm, we use parallel package to parallelize computations the factorization function.

3. Parameter Tuning

We tested the following parameters here. f = 5, 10, 15 lambda_als = 1,5,10 lambda_p = 1, 5, 10 sigmas = 1,2,3

```
fs <- c(5, 10, 15)
lambdas_als <- c(1, 5, 10)
lambdas_P <- c(1, 5, 10)
sigmas <- c(1, 2, 3)

cv.df <- expand.grid("fs"=fs,
                     "lambdas_als"=lambdas_als,
                     "lambdas_p"=lambdas_P,
                     "sigmas"=sigmas)
```

The code of parameter tuning is

```
cl <- makeCluster(cores)

x <- cv.df %>%
  as_tibble() %>%
  mutate(train_mean=NA,
         test_mean=NA)

for (i in 1:dim(x)[1]){
  tmp <- cv.functionYQ(dat_train=data,
                       K=5,
                       f=x[i,]$fs,
                       maxIter=15,
                       lambdas_als=x[i,]$lambdas_als,
                       lambdas_p=x[i,]$lambdas_p,
                       sigmas=x[i,]$sigmas)

  x[i,"train_mean"]=tmp$mean_train_rmse
  x[i,"test_mean"]=tmp$mean_test_rmse

  rm(tmp)
}
save(x, file="../output/cvtmp.RData")

stopCluster(cl)
```

The result is like

```
load(file="../output/cvtmp.RData")
x
```

```
## # A tibble: 81 x 6
##       fs lambdas_als lambdas_p sigmas train_mean test_mean
##    <dbl>       <dbl>     <dbl>  <dbl>      <dbl>     <dbl>
##  1     5           1         1      1      0.734     1.05
##  2    10           1         1      1      0.634     1.13
##  3    15           1         1      1      0.573     1.17
##  4     5           5         1      1      0.747     1.00
##  5    10           5         1      1      0.641     1.00
##  6    15           5         1      1      0.581     1.00
##  7     5          10         1      1      0.768     0.992
##  8    10          10         1      1      0.671     0.982
##  9    15          10         1      1      0.615     0.973
## 10     5           1         5      1      0.820     1.05
```

```
## # ... with 71 more rows
```

The optimal parameters for ALS + KRR is

```
x[which.min(x$test_mean), ][1:4]
```

```
## # A tibble: 1 x 4
##      fs lambdas_als lambdas_p sigmas
##   <dbl>       <dbl>     <dbl>  <dbl>
## 1    15          10         1      1
```

4. Optimal Model

- Model Fitting Here, we fit the ALS with post-processing, KRR

```
cl <- makeCluster(cores)

alsP3Result <- ALS_KRR(data, train, test, f=15, maxIters=10,
                       lambda_als=10, lambda_p=1, sigma=1)

stopCluster(cl)
save(alsP3Result, file = "../output/alsP3Result.RData")
```

- Model Evaluation The final result of our model 1 is

```
load(file="../output/alsP3Result.RData")
cat("RMSE for train set:", round(alsP3Result$train_RMSE, 4),
    "  RMSE for test set:", round(alsP3Result$test_RMSE, 4))
```

```
## RMSE for train set: 0.6216   RMSE for test set: 0.9698
```

#Model 2:Alternating Least Squares + Penalty of magnitudes + Temporal Dynamics + Kernel Ridge Regression

Step 1. Load Data and Train-Test Split

Step 2. Model Setup

1. 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

- 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.

- Temporal Dynamics

$$\hat{r}_{ui} = q_i^T p_u + \mu + b_u + b_i + \alpha_u dev_u(t)$$

The $dev_u(t)$ is the related to the difference between rating time of user and their average rating time, measured by

$$dev_u(t) = sign(t - t_u)\mathring{\text{u}}|t - t_u|^\beta$$

The final objective function of our model 2 is

$$min_{q,p} \sum_{(u,i)\in K} [r_{ui} - (q_i^T p_u + \mu + b_u + b_i + \alpha_u dev_u(t)]^2 + \lambda \sum (\|q_i\|^2 + \|p_u\|^2 + b_i^2 + b_u^2 + \alpha_u^2)$$

2. Algorithm Setup

For matrix factorization, with the method in paper4, to include b_u, b_i, and a_u in the iterations, we add one row to the movie matrix and two rows to the user matrix. To speed up the algorithm, we use parallel package to parallelize computations and tibble in the factorization function.

3. Parameter Tuning

To save the time, with the refult of parameters tuning of ALS and KRR from the former model and some attempts, we only tested the following parameters here. lambda_als = 1,5,10 beta = 0.2,0.4,0.5 And set f = 15 lambda_p = 1 sigmas = 1

```
fs <- c(15)
lambdas_P <- c(1)
sigmas = c(1)


lambdas_als <- c(1, 5,10)
betas <- c(0.2,0.4,0.5)

cv.df <- expand.grid("fs"=fs,
                     "lambdas_als"=lambdas_als,
                     "lambdas_p"=lambdas_P,
                     "sigmas" =sigmas,
                     "betas"=betas)
```

The code of parameter tuning is

```
set.seed(1)
cl <- makeCluster(cores)

arp_result=cv.df %>%
  as_tibble() %>%
  mutate(train_mean=NA,test_mean=NA)

for (i in 1:dim(arp_result)[1]){

  tmp <- cv.functionA3R3P3(dat_train=data,
```

```
                        K=4,
                        f=x[i,]$fs,
                        maxIter=15,
                        lambdas_als=x[i,]$lambdas_als,
                        lambdas_p=x[i,]$lambdas_p,
                        sigmas=x[i,]$sigmas,
                        betas=x[i,]$betas)

  arp_result[i,"train_mean"]=tmp$mean_train_rmse
  arp_result[i,"test_mean"]=tmp$mean_test_rmse

  rm(tmp)
}

save(arp_result, file="../output/cvARP.RData")

stopCluster(cl)
```

The result is

```
load(file="../output/cvARP.RData")
arp_result
```

```
## # A tibble: 9 x 7
##      fs lambdas_als lambdas_p sigmas betas train_mean test_mean
##   <dbl>       <dbl>     <dbl>  <dbl> <dbl>      <dbl>     <dbl>
## 1    15           1         1      1   0.2      0.497      1.25
## 2    15           5         1      1   0.2      0.469      1.32
## 3    15          10         1      1   0.2      0.485      1.30
## 4    15           1         1      1   0.4      0.889      1.33
## 5    15           5         1      1   0.4      0.929      1.44
## 6    15          10         1      1   0.4      0.935      1.43
## 7    15           1         1      1   0.5      1.17       1.48
## 8    15           5         1      1   0.5      1.25       1.62
## 9    15          10         1      1   0.5      1.25       1.61
```

The optimal parameters for ALS + R1R3 + KRR is

```
arp_result[which.min(arp_result$test_mean),][1:5]
```

```
## # A tibble: 1 x 5
##      fs lambdas_als lambdas_p sigmas betas
##   <dbl>       <dbl>     <dbl>  <dbl> <dbl>
## 1    15           1         1      1   0.2
```

4. Optimal Model

- Model Fitting Here, we fit the complete model with post-processing, KRR

```
cl <- makeCluster(cores)
ALS.R1R3.KRR.model <- ALS_R_KRR(data, data_train, data_test, f = 15,
                                maxIters = 10, lambda_als = 1, lambda_p = 1,
                                sigma = 1, beta = 0.2)
stopCluster(cl)
save(ALS.R1R3.KRR.model, file="../output/ALS_R1R3_KRR_model.RData")
```

```
stopCluster(cl)
```

- Model Evaluation We can see that when we add post-processing to the matrix factorization, the performance decrease. The final result of our model 2 is

```
load(file="../output/ALS_R1R3_KRR_model.RData")
cat("RMSE for train set:", round(ALS.R1R3.KRR.model$train_RMSE, 4),
    "  RMSE for test set:", round(ALS.R1R3.KRR.model$test_RMSE, 4))
```

```
## RMSE for train set: 0.5002   RMSE for test set: 1.2418
```

We noticed that without P3, the matrix factorization alone performs better. With our previous attempts, we chose the following parameters. (We also noticed that the optimal parameters for matrix factorization are different with or without the post-processing)

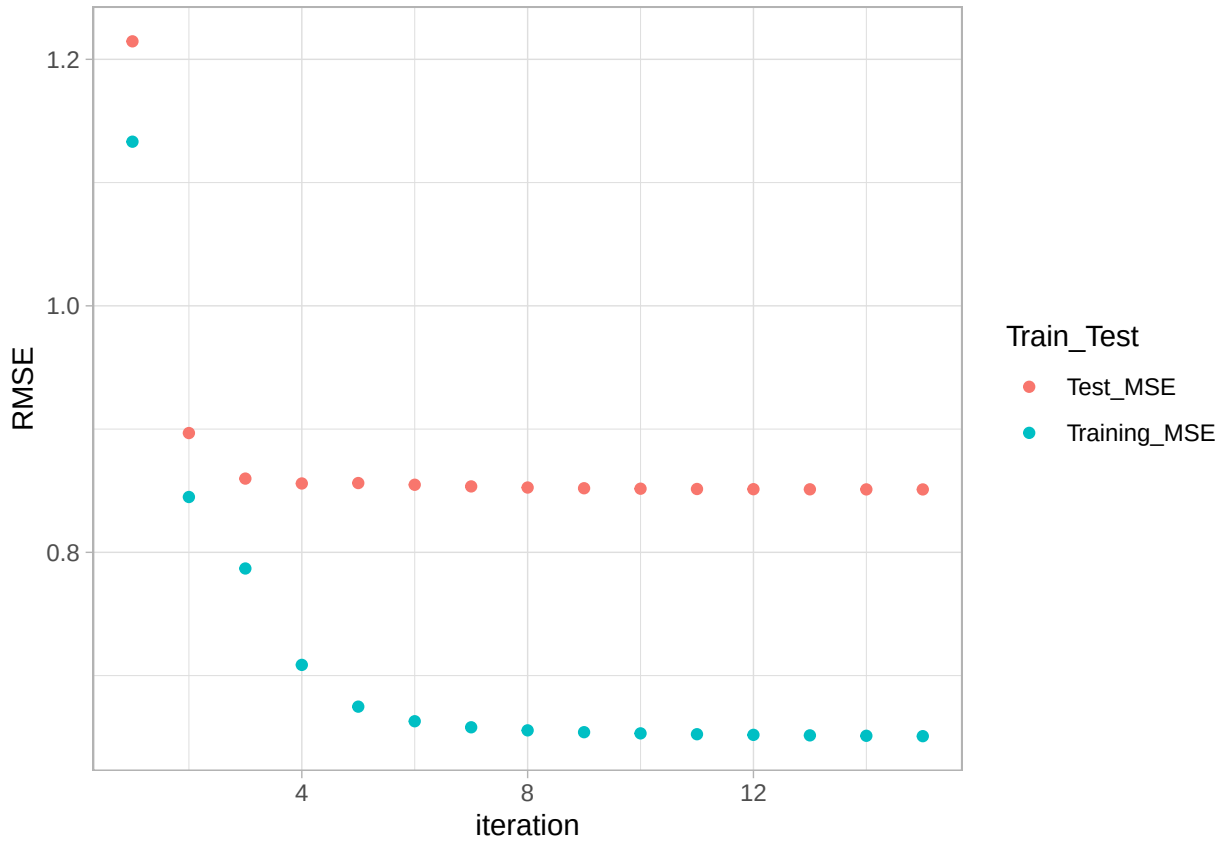
```
set.seed(1)
cl <- makeCluster(cores)

ALS.R1R3.only <- ALS.R1R3(f = 10, lambda = 10, beta = 0.4, maxIters = 15 , data = data,
                          train = data_train, test = data_test)

stopCluster(cl)
save(ALS.R1R3.only, file="../output/ALS_R1R3_only.RData")
```

Visualization of training and test RMSE by different iterations

```
load(file="../output/ALS_R1R3_only.RData")
g2 <- data.frame(iteration = c(1:15),Training_MSE = ALS.R1R3.only$`Train RMSE`,
                 Test_MSE = ALS.R1R3.only$`Test RMSE`) %>%
  gather(key = Train_Test,value = RMSE,-iteration) %>%
  ggplot(aes(x = iteration, y = RMSE, col = Train_Test)) + geom_point() +
  theme_light()
g2
```



We can see, without post-processing, the regularization can significantly reduce the RMSE in the second iteration. The final result is

```
load(file="../output/ALS_R1R3_only.RData")
cat("RMSE for train set:", round((ALS.R1R3.only$`Train RMSE`)[15], 4),
    "  RMSE for test set:", round((ALS.R1R3.only$`Test RMSE`)[15], 4))
```

```
## RMSE for train set: 0.6509   RMSE for test set: 0.851
```

Summary

	A3 + KRR	A3 + R1R3 + KRR	A3 + R1R3
training RMSE	0.6216	0.5002	0.6509
test RMSE	0.9698	1.2418	0.851

By comparing RMSEs, we know that, given (A3+P3), adding (R1+R3) tends to overfit. With (R1+R3), the training RMSE is lower but the test RMSE is higher. But without KRR, (A3 + R1R3) performs better than (A3 + KRR). (A3 + R1R3) is the most robust model, with a higher training MSE but lower test RMSE. Thus, we think KRR and R1R3 may not work well together, but can improve the original matrix factorization separately.