Project4

In this project we want to implement, evaluate and compare two Collaborative Filtering algorithms - Stochastic Gradient Descent given kernel redge regression, and Probabilistic Stochastic Gradient Descent given kernel redge regression.

Step 1 Load Data and Train-test Split

This dataset that we used describes 5-star rating of 9742 movies from 610 users. Each user and each movie are represented by an id number.

```
data <- read.csv("../data/ml-latest-small/ratings.csv")
set.seed(0)
test_idx <- sample(1:nrow(data), round(nrow(data)/5, 0))
train_idx <- setdiff(1:nrow(data), test_idx)
data_train <- data[train_idx,]
data_test <- data[test_idx,]
```

Step 2 Matrix Factorization

Step 2.1 Stochastic Gradient Descent given Kernel Ridge Regression

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

The first matrix factorization method is Stochastic Gradient Descent.
f: dimension of latent factors.
$q_i \in R^f$:factors related to the movie i.
$p_u \in R^f$: factors related to the user u.

The objective function for Stochastic Gradient Descent is:

$$min_{q^*p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

We will using the cross validation later to get the best parameter f, and using the function "gradesc" to get the final ans p, q, train_RMSE , test_RMSE.

Minimizing the objective function:
- Define f = dimension of latent factors, U = number of unique user id, and I = number of unique movie id, $lambda$ = regularization parameter
- Random assign values to a $f * I$ matrix q and $f * U$ matrix p
- For $t = 0, 1, ..., max.iteration$:
1. Calculate the prediction error $e_{ui} = r_{ui} - q_i^T p_u$ 2. Update $q_i = q_i + \gamma(e_{ui}p_u - \lambda q_i)$
Update $p_u = p_u + \gamma(e_{ui}q_i - \lambda p_u)$
3. Update q_i and p_u until we reach the max.iteration

```
U <- length(unique(data$userId))
I <- length(unique(data$movieId))
source("../lib/Matrix_Factorization_A1.R")
```

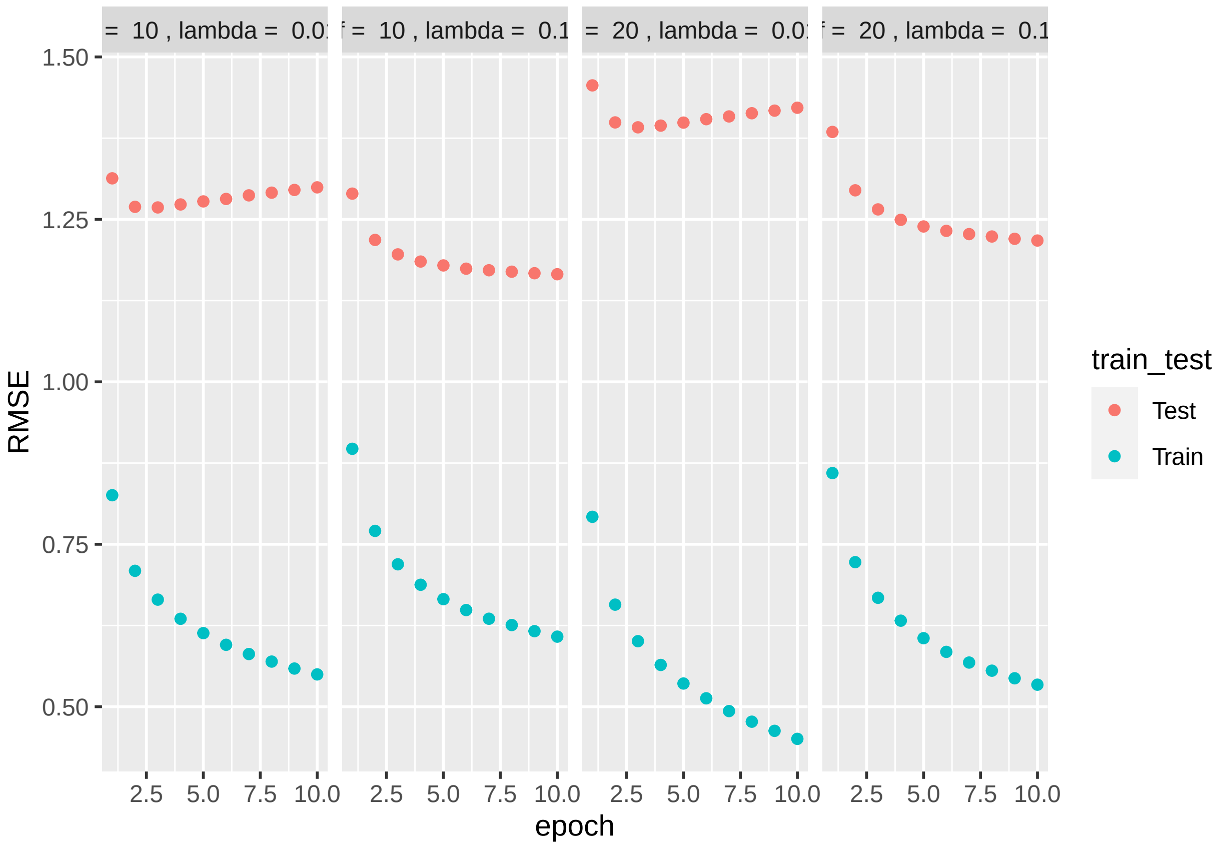
Step 2.1.1 Parameter Tuning

According to the cross validation, the best tunning parameter is $f = 10$, $\lambda = 0.1$

```
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)
```

```
result_summary_A1 <- array(NA, dim = c(nrow(f_l), 10, 4))
run_time_A1 <- 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_A1 <- cv.function(data_train, K = 5, f = f_l[i,1], lambda = 10^f_l[i,2])
    result_summary_A1[,,i] <- matrix(unlist(current_result), ncol = 10, byrow = T)
    print(result_summary_A1)

})

save(result_summary_A1, file = "../output/rmse_A1.Rdata")
```

```
load("../output/rmse_A1.Rdata")
rmse <- data.frame(rbind(t(result_summary_A1[1,,]), t(result_summary_A1[2,,])), train_test = rep(c("Tra
  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)
```



```
rmse %>% filter(train_test == 'Test') %>% group_by(par) %>% summarise(mean = mean(RMSE)) %>% arrange(mea
## # A tibble: 4 x 2
##   par                         mean
##   <fct>                      <dbl>
## 1 f =  10 , lambda =  0.1    1.19
## 2 f =  20 , lambda =  0.1    1.26
## 3 f =  10 , lambda =  0.01   1.29
## 4 f =  20 , lambda =  0.01   1.41
```

Step 2.1.2 Using the parameter: f=10, lambda=0.1

```r
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_A1.RData")
```

Step 2.1.3 P3 Postprocessing

After matrix factorization, postporcessing will be performed to improve accuracy.

P3:Postprocessing SVD with kernel ridge regression Section 3.6

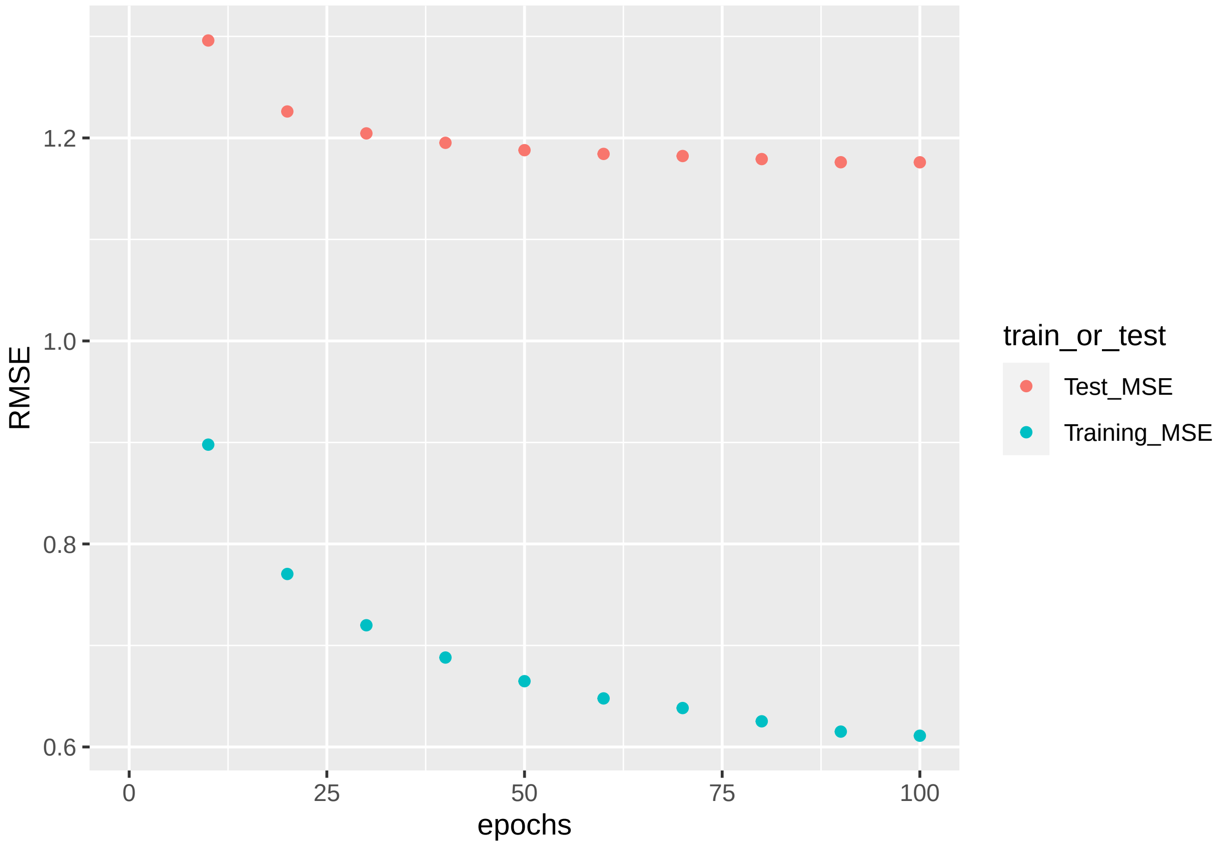
```r
source('../lib/Post_Process_P3.r')
```

```
## Loading required package: pracma

##
## Attaching package: 'pracma'

## The following object is masked from 'package:purrr':
##
##     cross
```

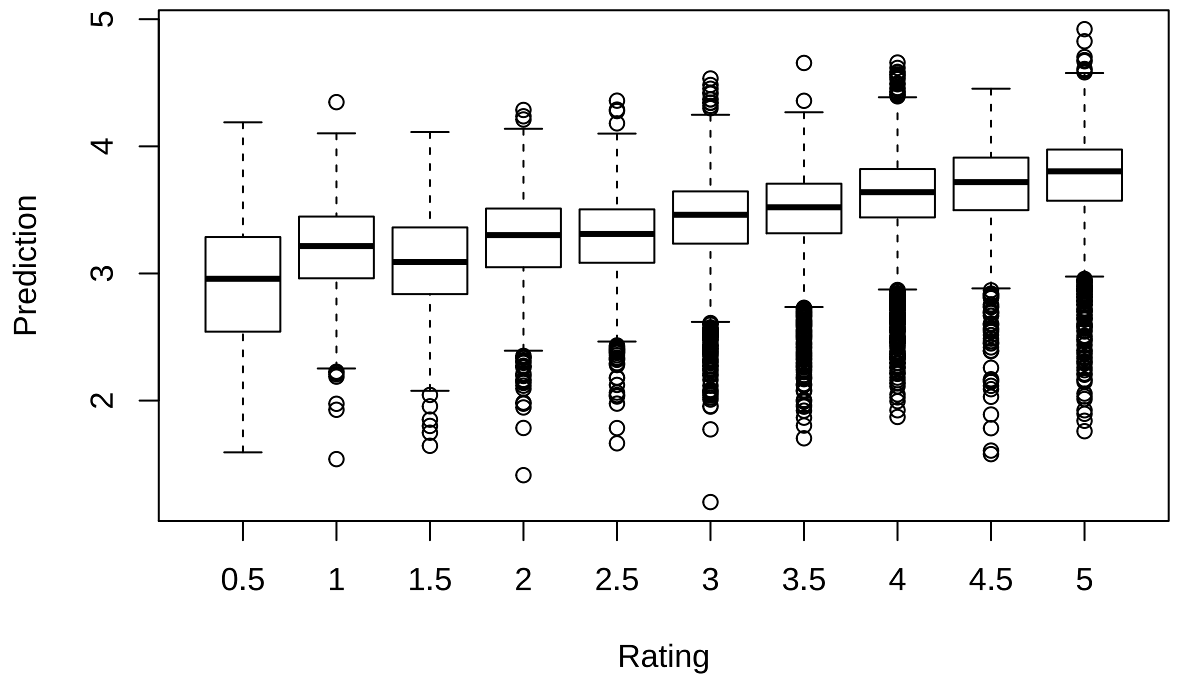
```r
load(file = "../output/mat_fac_A1.RData")

pred_rating_A1 <- t(result$q) %*% result$p
X <- X_mat(result$q)
n <- nrow(X)
lambda <- 0.5

#A1_P3_rating=svd_krr(n=n,lambda=lambda,X=X,y=pred_rating_A1)
#save(A1_P3_rating,file='../output/A1_P3_rating.RData')

load('../output/A1_P3_rating.RData')
#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_A1 <- apply(data_test, 1, extract_pred_rating, A1_P3_rating)
```

Step 2.1.4 Visualize training and testing RMSE by different epochs

```r
RMSE <- data.frame(epochs = seq(10, 100, 10), Training_MSE = result$train_RMSE, Test_MSE = result$test_

RMSE %>% ggplot(aes(x = epochs, y = RMSE,col = train_or_test)) + geom_point() + scale_x_discrete(limits
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```



```r
rmse_mat=function(P,Y){
  return (sqrt(mean((P-Y)^2)))
}

#mean(P)
pred_mean_A1 <- mean(pred_test_rating_A1)
#mean(test)
mean_test_rating <- mean(data_test$rating)

#mean(test) - mean(P)
mean_diff_A1 <- mean_test_rating - pred_mean_A1

data_test$pred_A1 <- pred_test_rating_A1
data_test$pred_adj1 <- pred_test_rating_A1 + mean_diff_A1

boxplot(data_test$pred_adj1 ~ data_test$rating, xlab = 'Rating', ylab = 'Prediction')
```



```r
#calculate RMSE
rmse_a1=rmse_mat(P=pred_rating_A1,Y=data_test$rating)
rmse_a1_p3=rmse_mat(P=A1_P3_rating,Y=data_test$rating)

cat("The RMSE of the adjusted model changes from", rmse_a1, ' to ',rmse_a1_p3)
```

```
## The RMSE of the adjusted model changes from 1.922347  to  1.465922
```

Step 2.2 PMF given Kernel Ridge Regression

A2. Gradient Descent with Probabilistic Assumptions Section 2

The second matrix factorization method is Gradient Descent with Probabilistic Assumptions.
f: dimension of latent factors.
$q_i \ in \ R^f$: factors related to the movie i.
$p_u \ in \ R^f$: factors related to the user u.

The objective function for Gradient Descent with Probabilistic Assumptions is:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_u}{2}\sum_{i=1}^{N}||U_i||_{Fro}^2 + \frac{\lambda_v}{2}\sum_{j=1}^{M}||V_j||_{Fro}^2$$

We will using the cross validation later to get the best parameter f, and using the function "pmf" to get the final ans p, q, train_RMSE , test_RMSE.

$U_i in R^{f} $: factor associated with user i, measure the extent of interest the user has in items that are high on the corresponding factors.
$V_j \ in \ R^f$: factor associated with item j, measure the extent to which the item possesses those factors.

Minimizing the objective function:
- Define f = dimension of latent factors, U = number of unique user id, I = number of unique movie id, $\lambda_u = \frac{\sigma^2}{\sigma_u^2}$, and $\lambda_v = \frac{\sigma^2}{\sigma_v^2}$
- Random assign values to a $f * I$ matrix U and $f * U$ matrix V
- For $t = 0, 1, ..., max.iteration$:

1. Calculate the prediction error $e_{ij} = R_{ij} - U_i^T V_j$
2. Update $U_i = U_i + \gamma(e_{ij}U_i - \lambda_u U_i)$
Update $V_j = V_j + \gamma(e_{ij}V_j - \lambda_v V_j)$
3. Update U_i and V_j until we reach the max.iteration

```
# Call A2 function
source("../lib/Matrix_Factorization_A2.R")
```

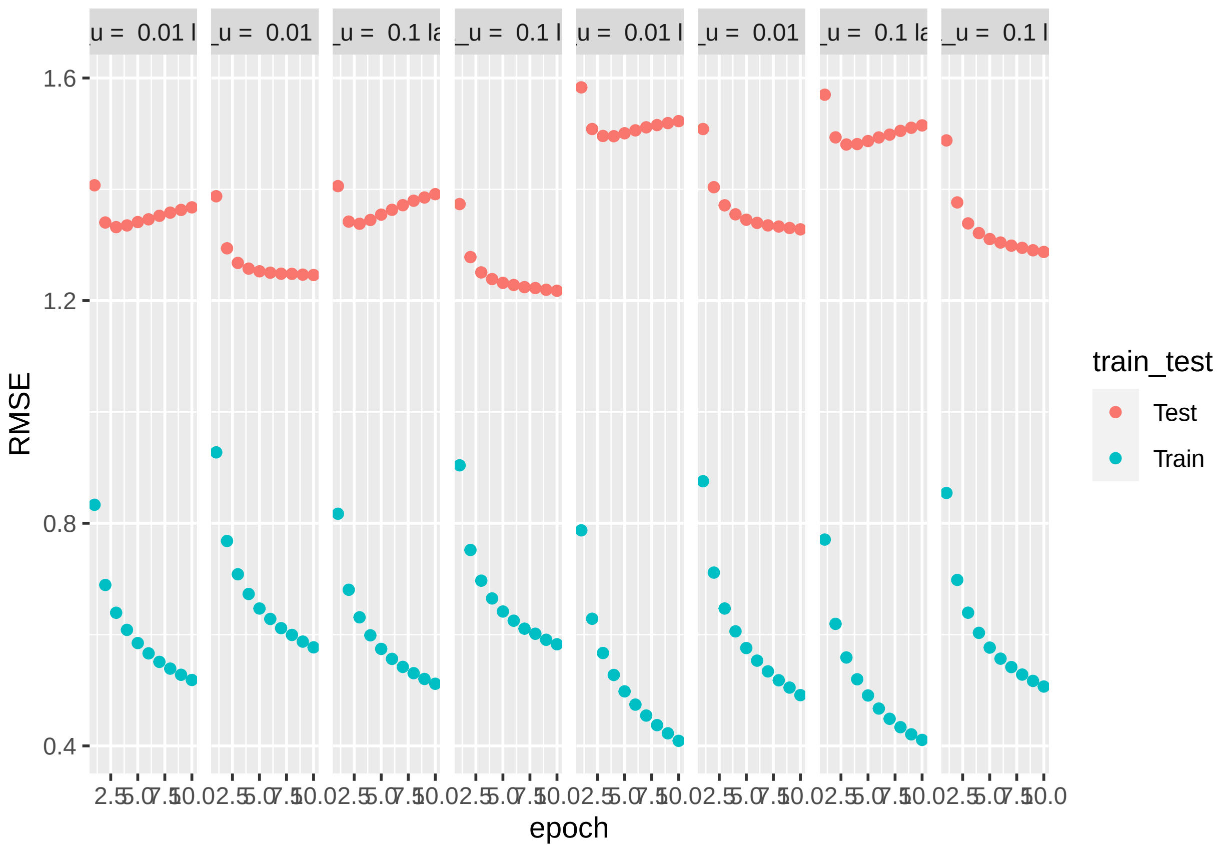
Step 2.2.1 Parameter Tuning

According to the cross validation, the best tunning patameterr is $f = 10$, $\lambda_u = 0.01$, $\lambda_v = 0.1$.

```
source("../lib/cross_validation_PMF.R")
f_list_A2 <- c(10, 20)
l_list_v <- c(0.01, 0.1)
l_list_u <- c(0.01, 0.1)
f_l_A2 <- expand.grid(f_list_A2, l_list_v, l_list_u)
```

```
result_summary_A2 <- array(NA, dim = c(4, 10, nrow(f_l_A2)))
run_time_A2 <- system.time(for(i in 1:nrow(f_l_A2)){
    par <- paste("f = ", f_l_A2[i,1], ", lambda = ", 10^f_l_A2[i,2])
    cat(par, "\n")
    current_result_A2 <- cv.function.pmf(data_train, K = 5, f = f_l_A2[i,1],lambda_v=f_l_A2[i,2], lambd
    result_summary_A2[,,i] <- matrix(unlist(current_result_A2), ncol = 10, byrow = T)
    print(result_summary_A2)
  })

save(result_summary_A2, file = "../output/rmse_A2.Rdata")
```

```
load("../output/rmse_A2.Rdata")
rmse_A2 <- data.frame(rbind(t(result_summary_A2[1,,]), t(result_summary_A2[2,,])),
                   train_test = rep(c("Train", "Test"), each = 8),
                   par = rep(paste("f = ", f_l_A2[,1], ", lambda_u = ", f_l_A2[,2], 'lambda_v = ', f_l_
  gather("epoch", "RMSE", -train_test, -par)

rmse_A2$epoch <- as.numeric(gsub("X", "", rmse_A2$epoch))
rmse_A2 %>% ggplot(aes(x = epoch, y = RMSE, col = train_test)) + geom_point() + facet_grid(~par)
```



```
rmse_A2 %>% filter(train_test == 'Test') %>% group_by(par) %>% summarise(mean = mean(RMSE)) %>% arrange
```

```
## # A tibble: 8 x 2
##   par                                               mean
##   <fct>                                            <dbl>
## 1 f =  10 , lambda_u =  0.1 lambda_v =  0.1         1.25
## 2 f =  10 , lambda_u =  0.01 lambda_v =  0.1        1.27
## 3 f =  20 , lambda_u =  0.1 lambda_v =  0.1         1.33
## 4 f =  10 , lambda_u =  0.01 lambda_v =  0.01       1.35
## 5 f =  20 , lambda_u =  0.01 lambda_v =  0.1        1.37
## 6 f =  10 , lambda_u =  0.1 lambda_v =  0.01        1.37
## 7 f =  20 , lambda_u =  0.1 lambda_v =  0.01        1.50
## 8 f =  20 , lambda_u =  0.01 lambda_v =  0.01       1.52
```

Step 2.2.2 Using the best parameter: f=10, lambda_u = 0.01, lambda_v = 0.1.

```
result_A2 <- pmf(f = 10, lambda_v = 0.1, lambda_u = 0.01,
                lrate = 0.01, max.iter = 100, stopping.deriv = 0.01,
                   data = data, train = data_train, test = data_test) ##f and lambda change

save(result_A2, file = "../output/mat_fac_A2.RData")
```

Step 2.2.3 P3 Postprocessing

After matrix factorization, postporcessing will be performed to improve accuracy.

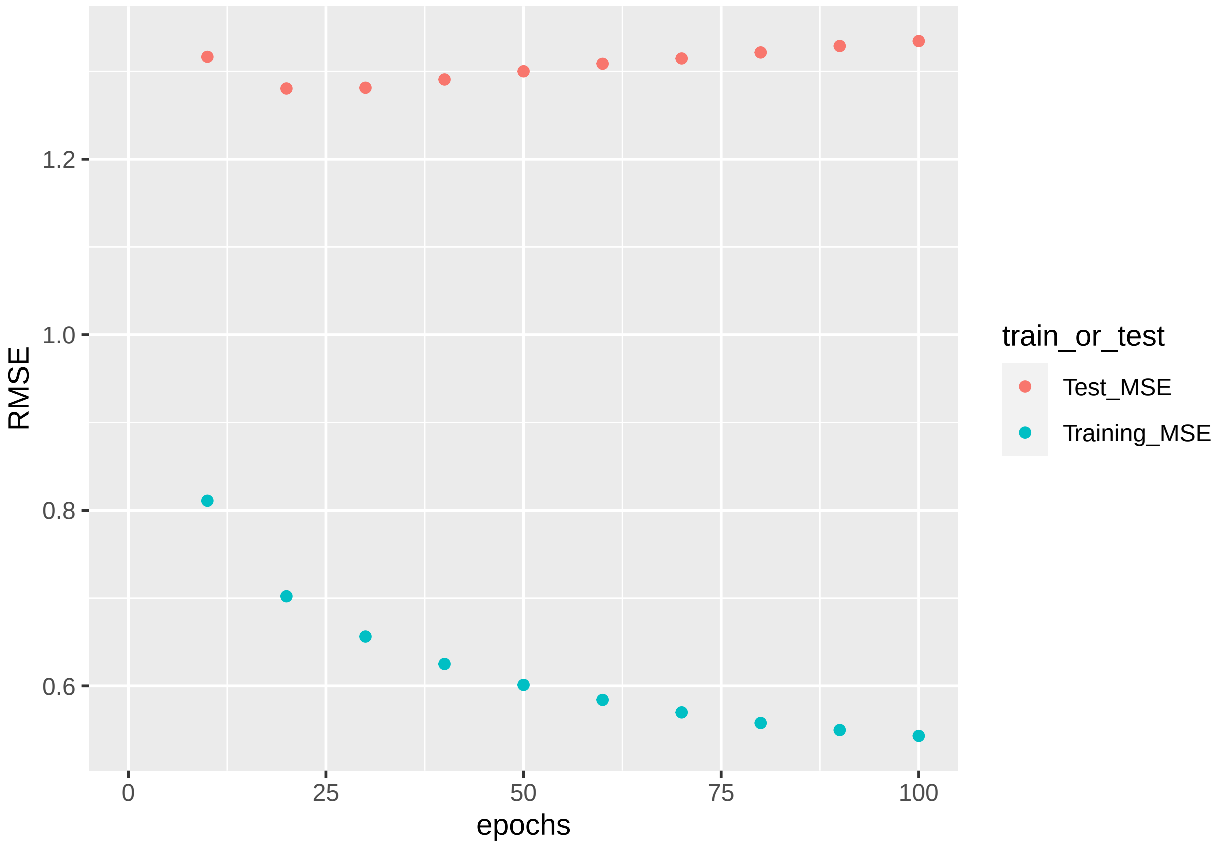
P3:Postprocessing SVD with kernel ridge regression Section 3.6

```r
source('../lib/Post_Process_P3.R')
load("../output/mat_fac_A2.RData")
pred_rating_A2=t(result_A2$q) %*% result_A2$p
X=X_mat(result_A2$q)
n=nrow(X)
lambda=0.5

#A2_P3_rating=svd_krr(n=n,lambda=lambda,X=X,y=pred_rating_A2)
#save(A2_P3_rating,file = '../output/A2_P3_rating.RData')

load('../output/A2_P3_rating.RData')
pred_test_rating_A2 <- apply(data_test, 1, extract_pred_rating, A2_P3_rating)
```

Step 2.2.4 visualize training and testing RMSE by different epochs

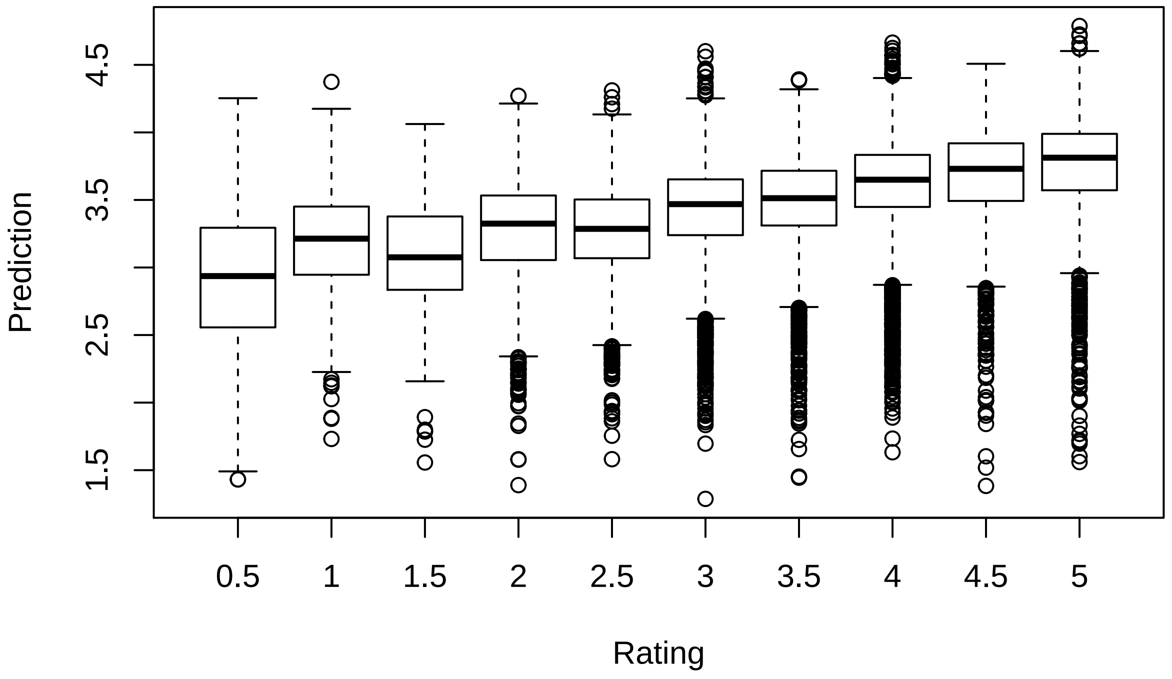
```r
RMSE_A2 <- data.frame(epochs = seq(10, 100, 10), Training_MSE = result_A2$train_RMSE, Test_MSE = result_

RMSE_A2 %>% ggplot(aes(x = epochs, y = RMSE,col = train_or_test)) + geom_point() + scale_x_discrete(limi
```

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



```r
#mean(P)
pred_mean_A2 <- mean(pred_test_rating_A2)
#mean(test)
mean_test_rating <- mean(data_test$rating)
```

```r
#mean(test) - mean(P)
mean_diff_A2 <- mean_test_rating - pred_mean_A2

data_test$pred_A2 <- pred_test_rating_A2
data_test$pred_adj2 <- pred_test_rating_A2 + mean_diff_A2

boxplot(data_test$pred_adj2 ~ data_test$rating, xlab = 'Rating', ylab = 'Prediction')
```



```r
#calculate RMSE
rmse_a2=rmse_mat(P=pred_rating_A2,Y=data_test$rating)
rmse_a2_p3=rmse_mat(P=A2_P3_rating,Y=data_test$rating)

cat("The RMSE of the adjusted model changes from", rmse_a2, ' to ',rmse_a2_p3)
```

```
## The RMSE of the adjusted model changes from 2.246795  to  1.398368
```

Summary

By comparison, the RMSE of Stochastic Gradient Descent given kernel ridge regression is 1.4659, while the RMSE of Probabilistic Matrix Factorization given kernel ridge regression is 1.3984. Therefore, Probabilistic Matrix Factorization given kernel ridge regression is better.