Project4

In this project, you are going to explore matrix factorization methods for recommender system. The goal is to match consumers with most appropriate products. Matrix factorization methods characterize both items and users by vectors of factors inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation. Matrix factorization generally has 3 parts:

- factorization algorithm
- regularization
- postpocessing

It is highly recommended to read this review paper.

Step 1 Load Data and Train-test Split

```
library(tidyverse)
library(ggplot2)
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 Algorithm and Regularization

For matrix factorization, we used an Alternating Least Squares algorithm, with Penalty of Magnitudes regularization. The referenced paper are:

A3. Alternating Least Squares Section 3.1

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

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

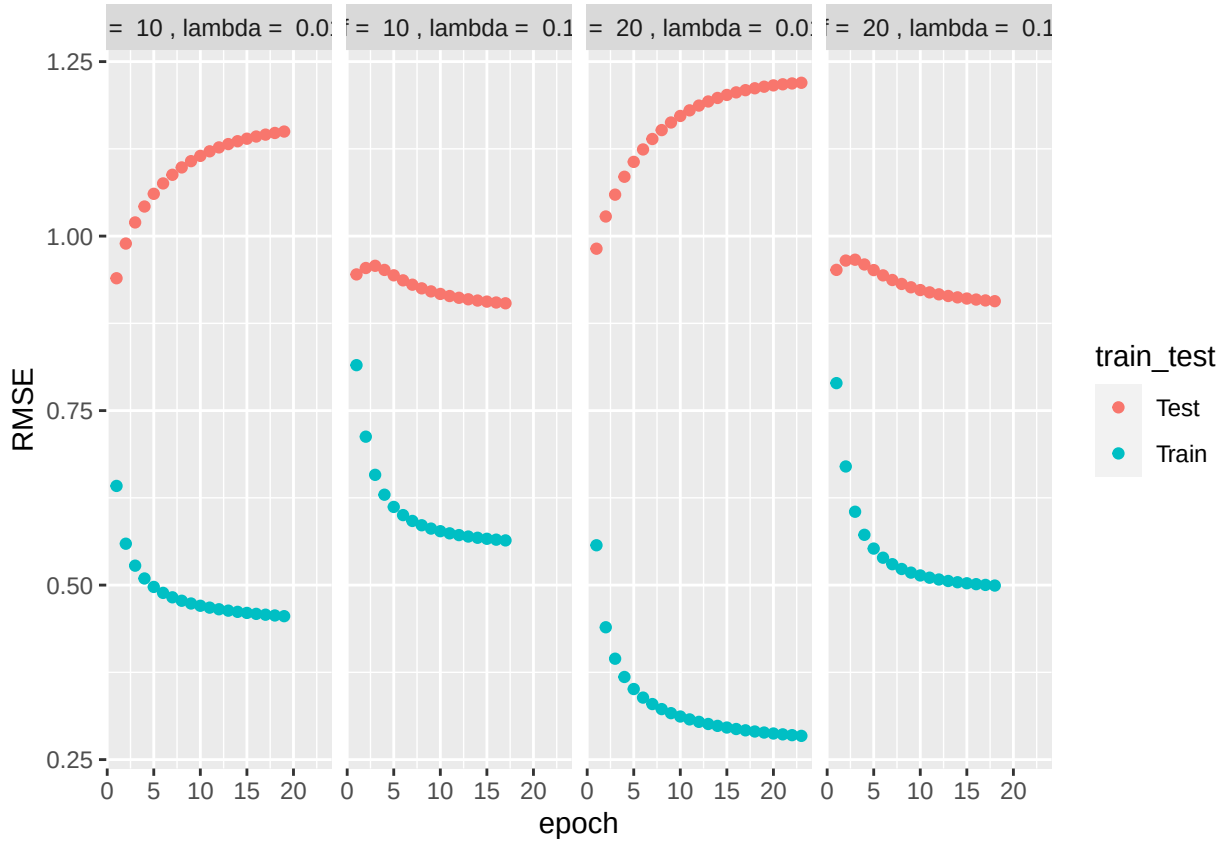
Step 2.2 Parameter Tuning Here we tune parameters, such as the dimension of factor and the penalty parameter λ by cross-validation.

```
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 <- array(NA, dim = c(nrow(f_l), 100, 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.als(data, K = 5, f = f_l[i,1], lambda = 10^f_l[i,2])
    result_summary[,,i] <- matrix(unlist(current_result), ncol = 100, byrow = T)
    save(result_summary, file = "../output/rmse_als.Rdata")

})
```

```
load("../output/rmse_als.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 %>%
  drop_na(RMSE) %>%
  ggplot(aes(x = epoch, y = RMSE, col = train_test)) + geom_point() + facet_grid(~par)
```



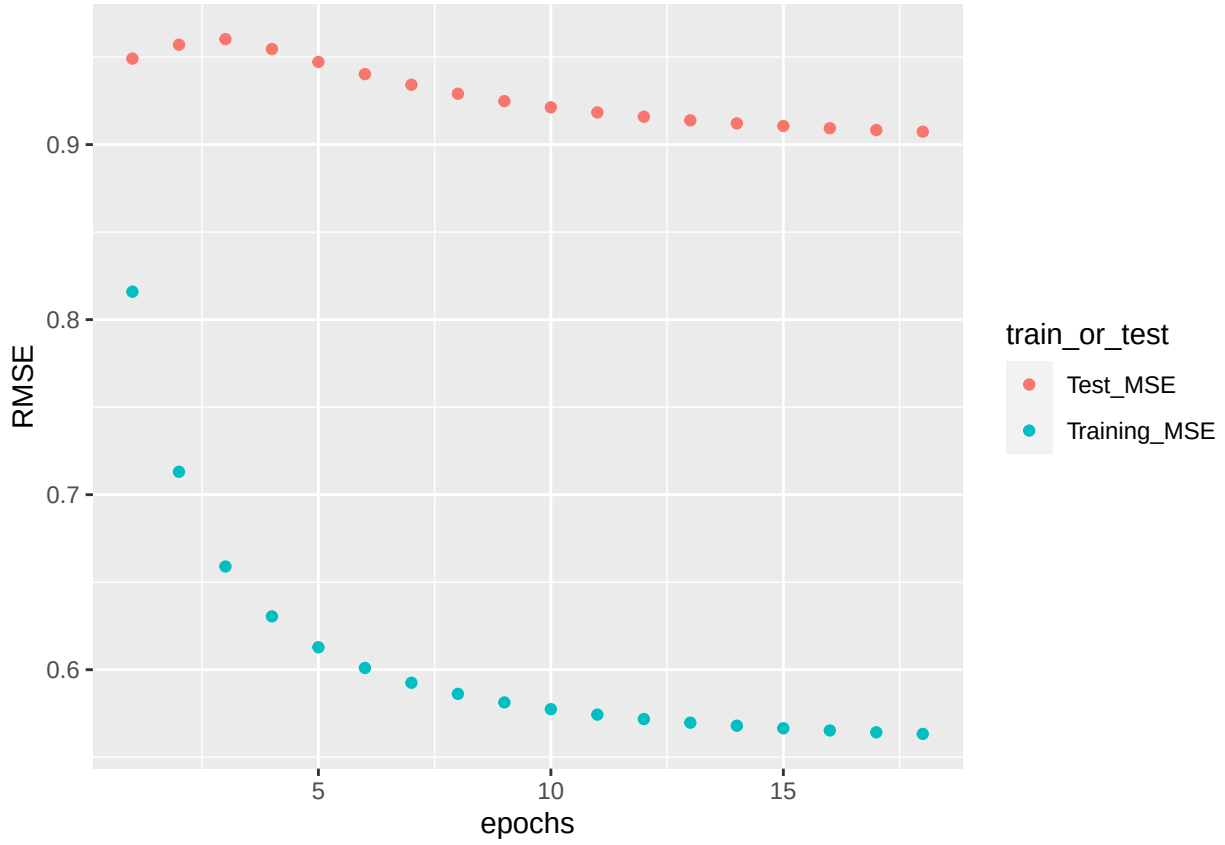

```
test_rmse <- rep(NA, 4)
for (i in 1:nrow(f_l)) {
  epoch_rmse <- result_summary[2,,i]
  epoch_rmse <- epoch_rmse[!is.na(epoch_rmse)]
  test_rmse[i] <- epoch_rmse[length(epoch_rmse)]
}
best_f <- f_l[which(test_rmse==min(test_rmse)), 1]
best_lambda <- 10**f_l[which(test_rmse==min(test_rmse)), 2]
```

Based on the cross-validation, the optimal values for the parameters (i.e. the lowest test error in cross-validation) are 10 factors, with lambda=0.1.

```
result <- als(f = best_f, lambda = best_lambda, max.iter = 100, stopping.thres = 0.001,
                  data = data, train = data_train, test = data_test)

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

Step 2.3 Evaluation on the Model without Postprocessing Here, we visualize training and testing RMSE by different epochs.

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

RMSE <- data.frame(epochs = seq(1, 100, 1), Training_MSE = result$train_RMSE,
                   Test_MSE = result$test_RMSE) %>%
  gather(key = train_or_test, value = RMSE, -epochs)

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




```
initial_RMSE <- RMSE %>% drop_na(RMSE) %>% pull(RMSE)
cat("The RMSE of the model before postprocessing is", initial_RMSE[length(initial_RMSE)],"\n")
```

```
## The RMSE of the model before postprocessing is 0.9073578
```

Step 3 Postprocessing

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

KNN The postprocessing method used here is SVD with KNN. The referenced paper is:

P2:Postprocessing SVD with KNN Section 3.5

Firstly, obtain the similarity data matrix and store it in RData file for further usage.

```
similarity.start <- proc.time()

m.s <- matrix(0,nrow = ncol(result$q), ncol = ncol(result$q))
for (i in 1:ncol(result$q)) {
  for (j in 1:ncol(result$q)) {
    m.s[i,j] = t(result$q[,i])%*%result$q[,j]/sqrt(sum(result$q[,i]^2))/sqrt(sum(result$q[,j]^2))
  }
}
colnames(m.s) <- colnames(result$q)
rownames(m.s) <- colnames(result$q)

similarity.stop <- proc.time()
similarity.time <- similarity.stop - similarity.start

save(m.s, file = "../output/movie_similarity.RData")
save(similarity.time, file = "../output/movie_similarity_time.RData")
```

Define the KNN_pred_rating function that will be used to derive the predicted rating for a given movie ID and user ID by using KNN method and the movie factor matrix obtained above.

```
load("../output/movie_similarity.RData")
load("../output/movie_similarity_time.RData")

pred_rating <- t(result$q) %*% result$p
k <- c(1,2,3,5,10,15)

#Define a function to extract the corresponding predicted rating by using KNN for the test set.
KNN_pred_rating <- function(test_set, pred, K){
  u <- test_set[1]
  m <- test_set[2]

  m_idx_urated <- unique(data$movieId[which(data$userId == u)])


  mid_ch <- c()
  for (i in 1:K) {
    m_idx <- ifelse(m %in% m_idx_urated,
      colnames(m.s)[which(m.s[as.character(m),] == sort(m.s[as.character(m),as.character(m_idx_urated)]
      decreasing = TRUE)[i+1])],
      colnames(m.s)[which(m.s[as.character(m),] == sort(m.s[as.character(m),as.character(m_idx_urated)]
      decreasing = TRUE)[i])])
    mid_ch <- c(mid_ch,m_idx)
  }

  pred_rating.um <- pred[mid_ch, as.character(u)]

  pred_rating <- mean(pred_rating.um)
  return(pred_rating)
}
```

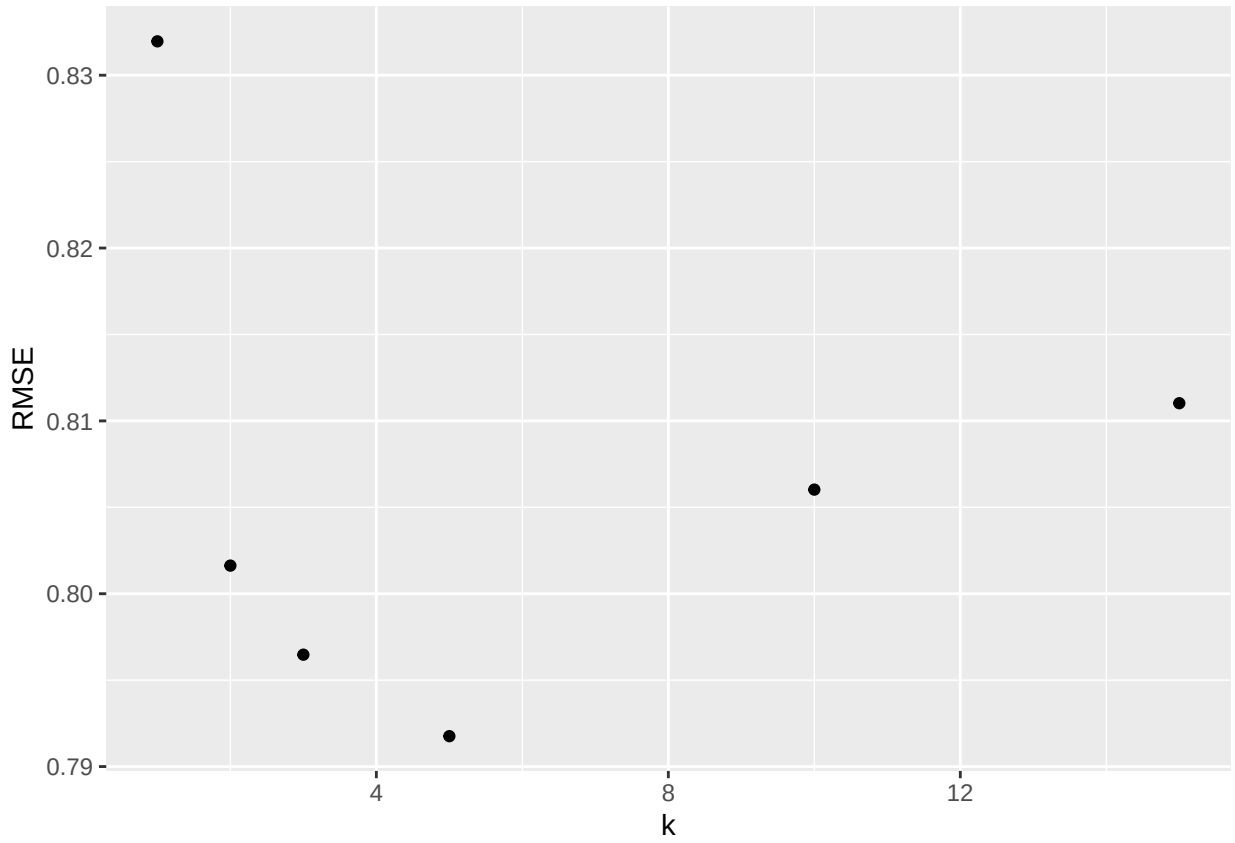
Use partial cross validation method to obtain an optimal k that would minimize the RMSE of the predicted rating by using the training set.

```
rmse_tr_te_k <- c()

k_cv <- 5
for (i in 1:length(k)) {
  rmse_tr_te_j <- c()
  for (j in 1:k_cv) {
    tr.te.idx <- sample(1:nrow(data_train),0.2*nrow(data_train),replace = TRUE)
    dt.tr.te <- data_train[tr.te.idx,]
    pred_tr_te_rating_k <- apply(dt.tr.te, 1, KNN_pred_rating, pred_rating,k[i])
    rmse_tr_te_j[j] <- sqrt(mean((dt.tr.te$rating - pred_tr_te_rating_k)^2))
  }
  rmse_tr_te_k[i] <- mean(rmse_tr_te_j)
  cat("The RMSE of the model for training cv using KNN postprocessing method while k = ",
      k[i]," is", rmse_tr_te_k[i],"\n")
}

save(rmse_tr_te_k, file = "../output/knn_cv_rmse.RData")
```

```
load("../output/knn_cv_rmse.RData")

for (i in 1:length(k)) {
  cat("The cross-validation RMSE of the model for KNN postprocessing method while k = ",
      k[i]," is", rmse_tr_te_k[i],"\n")
}
```

```
## The cross-validation RMSE of the model for KNN postprocessing method while k =  1  is 0.8319596
## The cross-validation RMSE of the model for KNN postprocessing method while k =  2  is 0.801627
## The cross-validation RMSE of the model for KNN postprocessing method while k =  3  is 0.7964731
## The cross-validation RMSE of the model for KNN postprocessing method while k =  5  is 0.791759
## The cross-validation RMSE of the model for KNN postprocessing method while k =  10  is 0.8060235
## The cross-validation RMSE of the model for KNN postprocessing method while k =  15  is 0.8110235
```

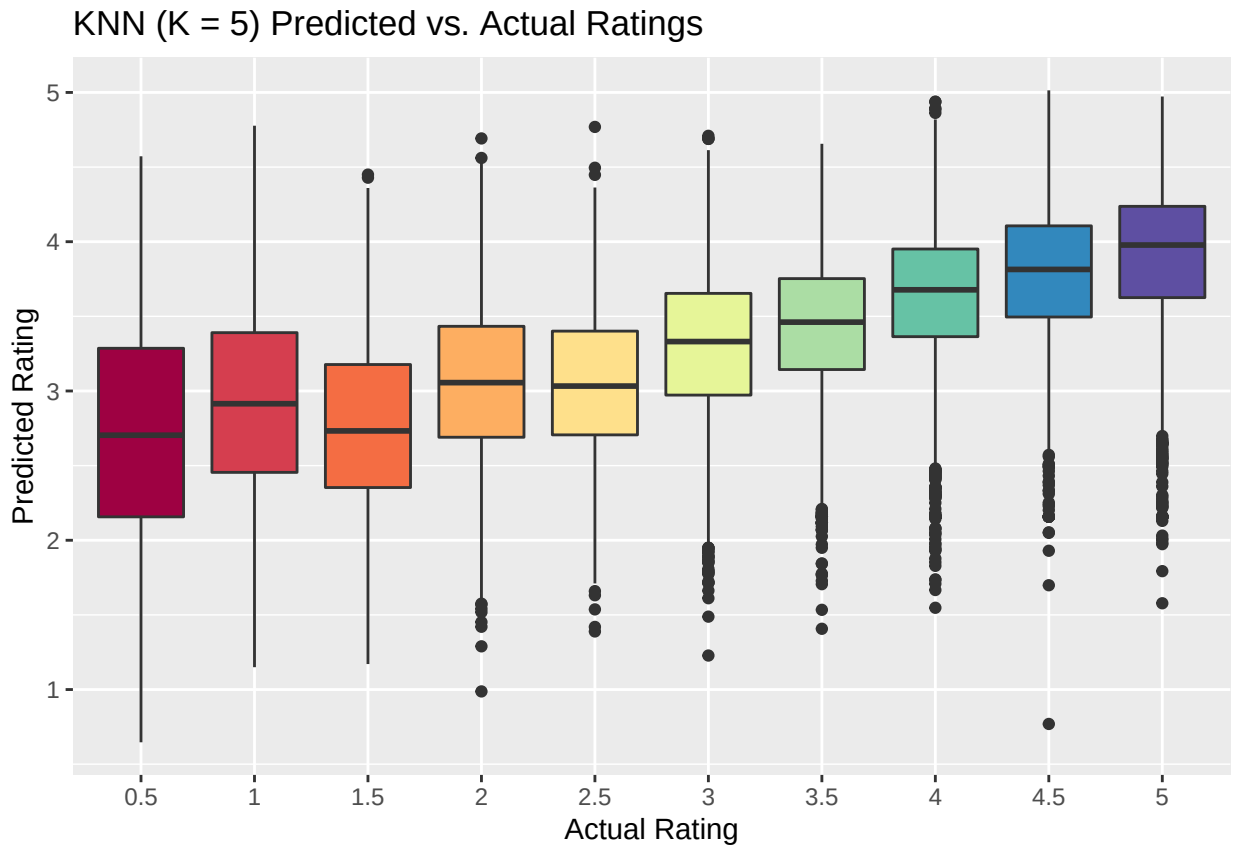
```
RMSE.df <- data.frame(k = c(1,2,3,5,10,15),
                      RMSE = rmse_tr_te_k)

ggplot(RMSE.df) +
  geom_point(aes(k, RMSE))
```




```
best_k <- k[which(rmse_tr_te_k==min(rmse_tr_te_k))]
```

Based on cross-validation, it turns out that k=5 would minimize the RMSE in predicting the movie ratings. Therefore, we choose k=5 as final parameter for KNN postprocessing method.

```
prediction.time <- system.time(pred_test_rating <- apply(data_test, 1, KNN_pred_rating,
                                                         pred_rating, best_k))

data_test$pred_knn <- pred_test_rating

# make into factor in order to allow boxplots to recognize rating as discrete and not continuous
data_test$rating <- as.factor(data_test$rating)

ggplot(data_test, aes(x=rating, y=pred_knn, fill=rating)) +
    geom_boxplot() +
    theme(legend.position="none") +
    ylab("Predicted Rating") +
    xlab("Actual Rating") +
    scale_fill_brewer(palette="Spectral") +
    ggtitle("KNN (K = 5) Predicted vs. Actual Ratings")
```




```
#calculate RMSE
rmse <- sqrt(mean((data_test$rating - data_test$pred_knn)^2))
```

```
## Warning in Ops.factor(data_test$rating, data_test$pred_knn): '-' not meaningful
## for factors
```

```
cat("The RMSE of the model with KNN is", rmse,"\n")
```

```
## The RMSE of the model with KNN is NA
```

```
cat("The time for calculating similarity is", similarity.time[1], "s\n")
```

```
## The time for calculating similarity is 969.35 s
```

```
cat("The time for making predictions is", prediction.time[1], "s")
```

```
## The time for making predictions is 113.31 s
```

Kernel Ridge Regression The postprocessing method used here is SVD with kernel ridge regression. The referenced paper is:

P3:Postprocessing SVD with kernel ridge regression Section 3.6

To improve efficiency, we will limit the ridge regression to 500 observations per user, taking the 500 most frequently-rated movies per user (as suggested in the reference paper). The select_movie() function below selects the list of rated movies to be used as ridge regression input for a given user.

```
topnum <- 500

movies_ranked_by_ratings <- data_train%>%
```

```
  count(movieId)%>%
  arrange(desc(n))%>%
  pull(movieId)

select_movie <- function(i){
  movies_i <- data_train%>%
      filter(userId == i)%>%
      pull(movieId)
  if (length(movies_i) <= topnum) {
    sel_list <- c(movies_i)
  }
  else {
    movies_i_ranked <- movies_ranked_by_ratings[movies_ranked_by_ratings %in% movies_i]
    sel_list <- movies_i_ranked[1:topnum]
  }
  return(sel_list)
}
```

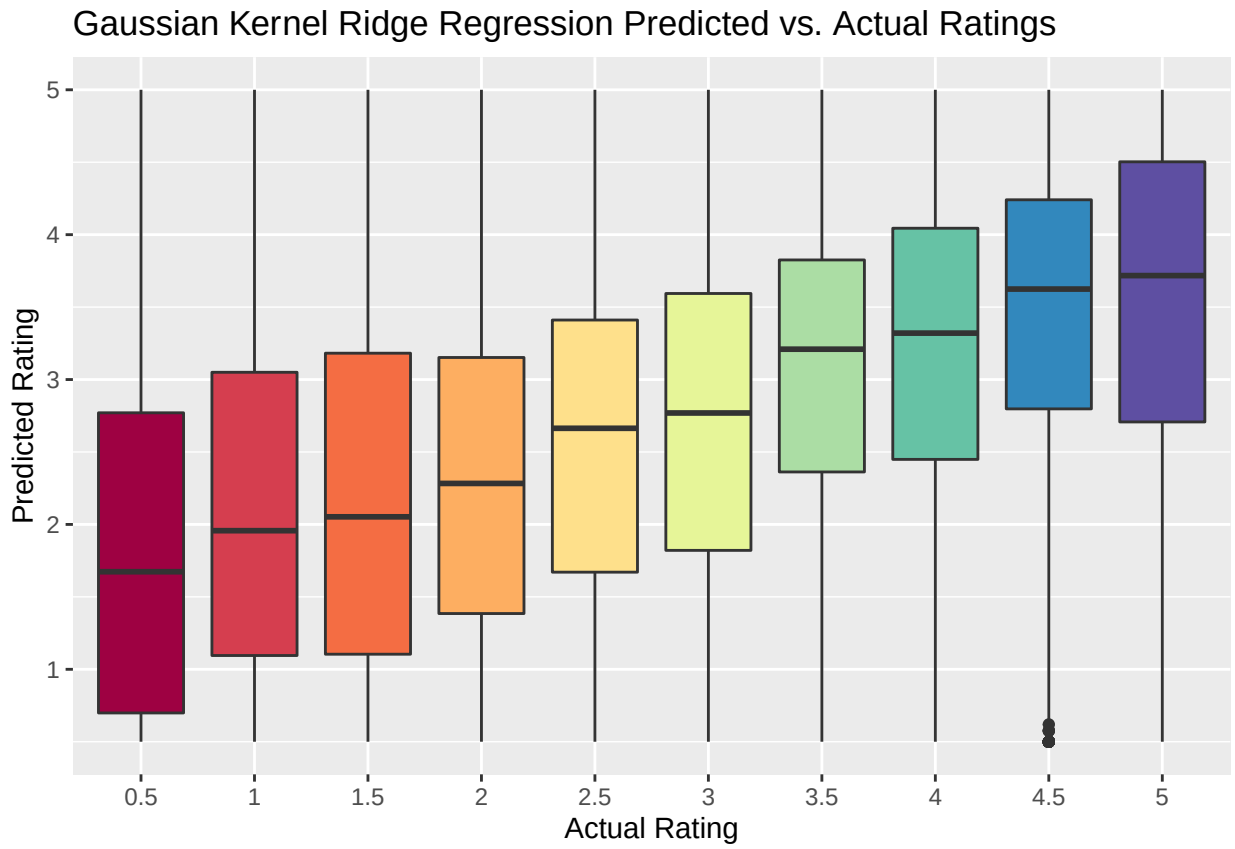
We explore kernel ridge regression using both a Gaussian kernel and a linear kernel.

Gaussian kernel For each user, we use the training data to fit a kernel ridge regression using a Gaussian kernel, to predict the ratings for the test set.

```
krr.gaussian.start <- proc.time()

lambda <- 0.5 #Tuning parameter
X_full <- t(scale(result$q, center=FALSE)) #Scaled observation matrix for every movie in the data set
test_preds <- data.frame(userId=c(), movieId=c(), pred_krr_gaussian=c())

for (i in unique(data_test$userId)){
  user_train_data <- data_train %>%
    filter(userId == i, movieId %in% select_movie(i))
  y <- user_train_data$rating
  X <- X_full[as.character(user_train_data$movieId), ]
  K <- exp(2 * (X %*% t(X) - 1))
  b <- solve(K + lambda * diag(nrow(K))) %*% y

  user_test_data <- data_test %>%
    filter(userId == i) %>%
    select(userId, movieId)
  X_test <- X_full[as.character(user_test_data$movieId), ]
  user_test_data$pred_krr_gaussian <- exp(2 * (X_test %*%t (X) - 1)) %*% b
  test_preds <- rbind(test_preds, user_test_data)
}

#Since ratings cannot be below 0.5 or above 5.0,
#we cap the predictions at these minimum and maximum values.
test_preds$pred_krr_gaussian <- ifelse(test_preds$pred_krr_gaussian < 0.5, 0.5,
    ifelse(test_preds$pred_krr_gaussian > 5, 5, test_preds$pred_krr_gaussian))
data_test <- data_test %>% left_join(test_preds, by=c('userId', 'movieId'))
rmse_KRR_gaussian <- sqrt(mean((data_test$rating - data_test$pred_krr_gaussian)^2))
```

```
## Warning in Ops.factor(data_test$rating, data_test$pred_krr_gaussian): '-' not
## meaningful for factors
```

```
krr.gaussian.stop <- proc.time()
krr.gaussian.time <- krr.gaussian.stop - krr.gaussian.start

ggplot(data_test, aes(x=rating, y=pred_krr_gaussian, fill=rating)) +
    geom_boxplot() +
    theme(legend.position="none") +
    ylab("Predicted Rating") +
    xlab("Actual Rating") +
    scale_fill_brewer(palette="Spectral") +
    ggtitle("Gaussian Kernel Ridge Regression Predicted vs. Actual Ratings")
```




```
cat("The RMSE of the model with Gaussian kernel ridge regression is", rmse_KRR_gaussian,"\n")
```

```
## The RMSE of the model with Gaussian kernel ridge regression is NA
```

```
cat("The time for training the model and making predictions is", krr.gaussian.time[1], "s")
```

```
## The time for training the model and making predictions is 11.47 s
```

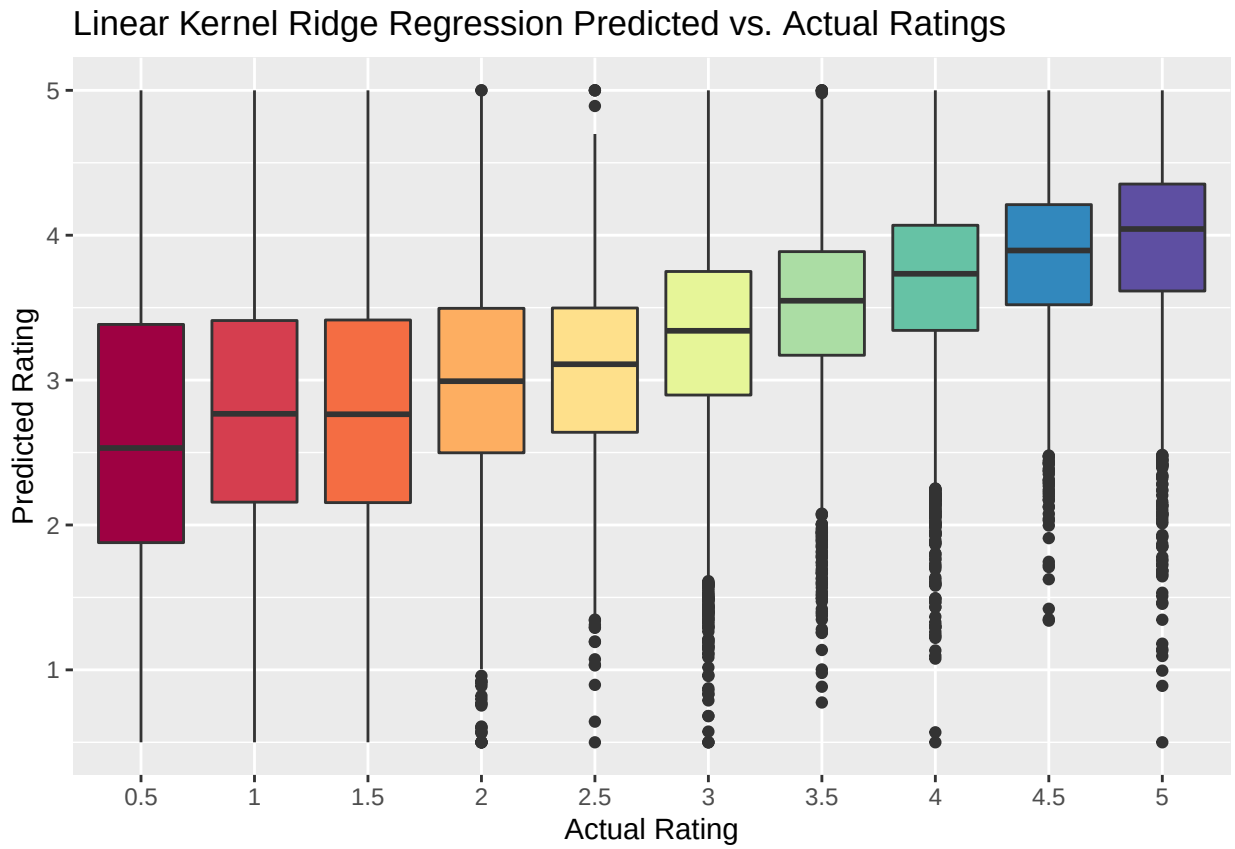
Linear kernel For each user, we use the training data to fit a kernel ridge regression using a linear kernel, to predict the ratings for the test set.

```
krr.linear.start <- proc.time()

lambda <- 0.5 #Tuning parameter
X_full <- t(scale(result$q, center=FALSE))
test_preds <- data.frame(userId=c(), movieId=c(), pred_krr_linear=c())
```

```r
for (i in unique(data_test$userId)){
  user_train_data <- data_train %>%
    filter(userId == i, movieId %in% select_movie(i))
  y <- user_train_data$rating
  X <- X_full[as.character(user_train_data$movieId), ]
  K <- X %*% t(X)
  b <- solve(K + lambda * diag(nrow(K))) %*% y

  user_test_data <- data_test %>%
    filter(userId == i) %>%
    select(userId, movieId)
 X_test <- X_full[as.character(user_test_data$movieId), ]
  user_test_data$pred_krr_linear <- X_test %*% t(X) %*% b
  test_preds <- rbind(test_preds, user_test_data)
}

#Since ratings cannot be below 0.5 or above 5.0,
#we cap the predictions at these minimum and maximum values.
test_preds$pred_krr_linear <- ifelse(test_preds$pred_krr_linear < 0.5, 0.5,
    ifelse(test_preds$pred_krr_linear > 5, 5, test_preds$pred_krr_linear))
data_test <- data_test %>% left_join(test_preds, by=c('userId', 'movieId'))
rmse_KRR_linear <- sqrt(mean((data_test$rating - data_test$pred_krr_linear)^2))
```

```
## Warning in Ops.factor(data_test$rating, data_test$pred_krr_linear): '-' not
## meaningful for factors
```

```r
krr.linear.stop <- proc.time()
krr.linear.time <- krr.linear.stop - krr.linear.start

ggplot(data_test, aes(x=rating, y=pred_krr_linear, fill=rating)) +
    geom_boxplot() +
    theme(legend.position="none") +
    ylab("Predicted Rating") +
    xlab("Actual Rating") +
    scale_fill_brewer(palette="Spectral") +
    ggtitle("Linear Kernel Ridge Regression Predicted vs. Actual Ratings")
```




```
cat("The RMSE of the model with linear kernel ridge regression is", rmse_KRR_linear,"\n")
```

```
## The RMSE of the model with linear kernel ridge regression is NA
```

```
cat("The time for training the model and making predictions is", krr.linear.time[1], "s")
```

```
## The time for training the model and making predictions is 10.45 s
```

Conclusion

Given matrix factorization based on Alternating Least Squares with Penalty of Magnitudes, the KNN model outperforms kernel ridge regression for postprocessing (regardless of the choice of Gaussian vs. linear kernel). The KNN model has a test RMSE of 0.8980, while the Gaussian and linear kernel ridge regressions have test RMSE of 1.4146 and 0.9477, respectively.

The KNN model does take more computation time, however, including over 16 minutes for measuring similarities between movies and between 2-3 minutes for making predictions. The kernel ridge regression models, on the other hand, each take less than half a minute to train the model and make predictions.