Project4

```
packages.used=c("dplyr", "tidyr","ggplot2","lsa","DT")
# check packages that need to be installed.
packages.needed=setdiff(packages.used,
                        intersect(installed.packages()[,1],
                                  packages.used))
# install additional packages
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}

if(!require("krr")){
  install.packages("remotes")
  remotes::install_github("TimothyKBook/krr")
}
```

```
library(dplyr)
library(tidyr)
library(ggplot2)
library(krr)
library(lsa)
library(DT)
```

Step 1 Load Data and Train-test Split

To make sure the training dataset has all userId and moiveId, we did a semi-random data split. First, generate a small dataset, which all userId and movieId appeared once. Then, randomly split the rest into training and testing. Finally, add the small dataset into the training dataset.

```
data <- read.csv("../data/ml-latest-small/ratings.csv")
set.seed(0)
## We want to make sure all movieId and all userId in the in the training dataset, we do
 a semi-random assignment

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

## split the remaining data into training and testing
rest <- shuffle.data[index,]
test_idx <- sample(rownames(rest), round(nrow(shuffle.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, shuffle.data[train_idx,])
data_test <- shuffle.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 (Alternating Least Squares) and Regularization (Penalty of Magnitudes + Bias and Intercepts)

```
U <- length(unique(data$userId))
I <- length(unique(data$movieId))
source("../lib/ALS.R1R2.R")
```

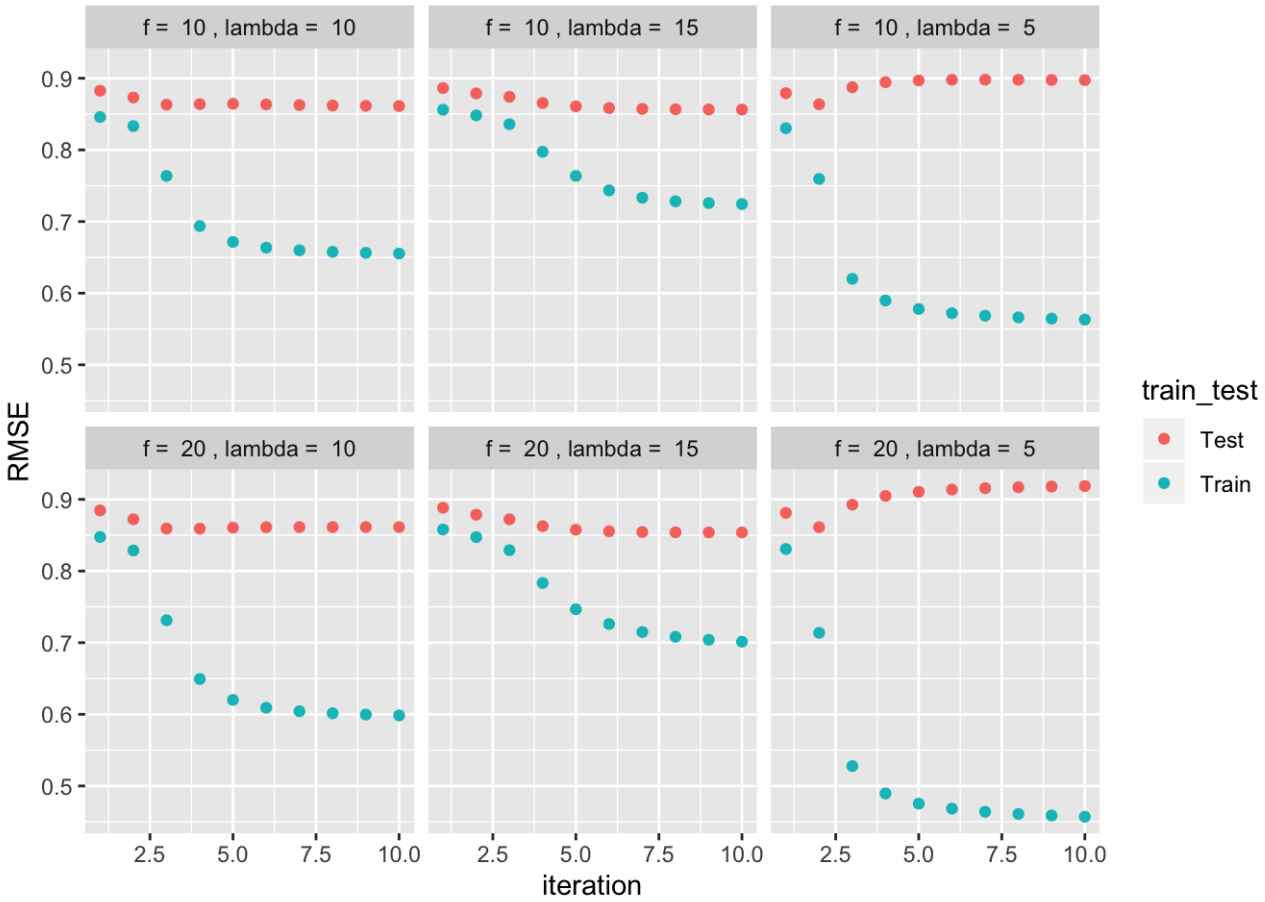
Step 2.2 Parameter Tuning (lambda and f latent variables)

```
source("../lib/ALS.Cross.Validation.R")
f_list <- c(10, 20)
l_list <- c(5, 10, 15)
f_l <- expand.grid(f_list, l_list)
```

```
# Cross validation takes around three hours, so don't run. Instead next code snippets loa
ds from memory
result_summary <- array(NA, dim = c(4, 10, nrow(f_l)))
ALS.CV.Runtime <- system.time(for(i in 1:nrow(f_l)){
    par <- paste("f = ", f_l[i,1], ", lambda = ", f_l[i,2])
    cat(par, "\n")
    current_result <- als.cv(data, K = 5, f = f_l[i,1], lambda = f_l[i,2])
    result_summary[,,i] <- matrix(unlist(current_result), ncol = 10, byrow = T)
    print(result_summary)

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

```
load("../output/alg.cv.rmse.Rdata")
rmse <- data.frame(rbind(t(result_summary[1,,]), t(result_summary[2,,])), train_test = re
p(c("Train", "Test"), each = 6), par = rep(paste("f = ", f_l[,1], ", lambda = ", f_l[,2
]), times = 2)) %>% gather("iteration", "RMSE", -train_test, -par)
rmse$iteration <- as.numeric(gsub("X", "", rmse$iteration))
rmse %>% ggplot(aes(x = iteration, y = RMSE, col = train_test)) + geom_point() + facet_wr
ap(~par, ncol=3)
```



1. There's not much difference between f=10 and f=20. Thus, we decide to go with the smaller number of latent varibles.
2. When lambda is small, the training RMSE drops fast, it is easy to overfit. When lambda = 10, the result is just right.
3. For f=10 and lambda=10, after 3 iterations, the test RMSE hardly changes. For the efficiency purpose, we decide to go with f=10, lambda=10, max.iter=3 for our best model.

ALS result

```
#Parameters for f, lambda, and max iter determined from image above of 5-fold cross valid
ation
result <- ALS.R1R2(f = 10, lambda = 10, max.iter=3, data=data, train=data_train, test=dat
a_test)
save(result, file = "../output/mat_fac.RData")
```

```
## The train RMSE for Alternating Least Squares + Penalty of Magnitudes and Bias and Inte
rcepts is 0.7665459 .
```

```
## The test RMSE for Alternating Least Squares + Penalty of Magnitudes and Bias and Inter
cepts is 0.8576189 .
```

Step 3 Postprocessing

1. SVD with KNN

First, generate a matrix of 9724*9724, which contains the pairwise cosine similarity of each movie using movie matrix q

```
#Generating the cosine similarity matrix can take 10 min, so we save as an Rdata after ge
nerating once
similarity.matrix <- cosine(result$q)
colnames(similarity.matrix) <- levels(as.factor(data$movieId))
rownames(similarity.matrix) <- levels(as.factor(data$movieId))
# save(similarity.matrix, file = "../output/similarity.matrix.Rdata")
```

Apply the one nearest neighbor method introduced in paper 2.

The knn prediction is the average rating of the most similar movie.

```
# load(file = "../output/similarity.matrix.Rdata")
source("../lib/KNN.Postprocessing.R")
KNN.result <- KNN.Post(data=data, train=data_train, test=data_test)
save(KNN.result, file = "../output/KNN.result.RData")
```

```
## The train RMSE for 1NN is 1.113356 .
```

```
## The test RMSE 1NN is 1.148548 .
```

The 1 nearest neighbor prediction does not improve the result, since it only takes movie matrix q into consideration.

Next, use the one nearest neighbor prediction as another predictor, together with the prediction rating from ALS, we fit a linear regression model to see whether it improves the prediction

```
load("../output/KNN.result.RData")
source("../lib/Getting.pred.R")


data_train$als.pred <- apply(data_train, 1, get.pred, est_rating=result$ALS.rating)
data_train$knn.pred <- apply(data_train, 1, get.pred, est_rating=KNN.result$knn.rating)

data_test$als.pred <- apply(data_test, 1, get.pred, est_rating=result$ALS.rating)
data_test$knn.pred <- apply(data_test, 1, get.pred, est_rating=KNN.result$knn.rating)

## fit linear model
als.knn.model <- lm(rating ~ als.pred+knn.pred, data=data_train)

## get training prediction
als.knn.train <- predict(als.knn.model, data_train[,c(5,6)])

## get testing prediction
als.knn.test <- predict(als.knn.model, data_test[,c(5,6)])

## get training and testing RMSE
als.knn.train.rmse <- sqrt(mean((data_train$rating - als.knn.train)^2))
als.knn.test.rmse <- sqrt(mean((data_test$rating - als.knn.test)^2))
```

```
## The train RMSE for 1NN after doing the linear regression with ALS is 0.746245 .
```

```
## The test RMSE 1NN after doing the linear regression with ALS is 0.8634571 .
```

The testing RMSE for linear regression of ALS+KNN is close to the ALS testing RMSE, but it is still higher.

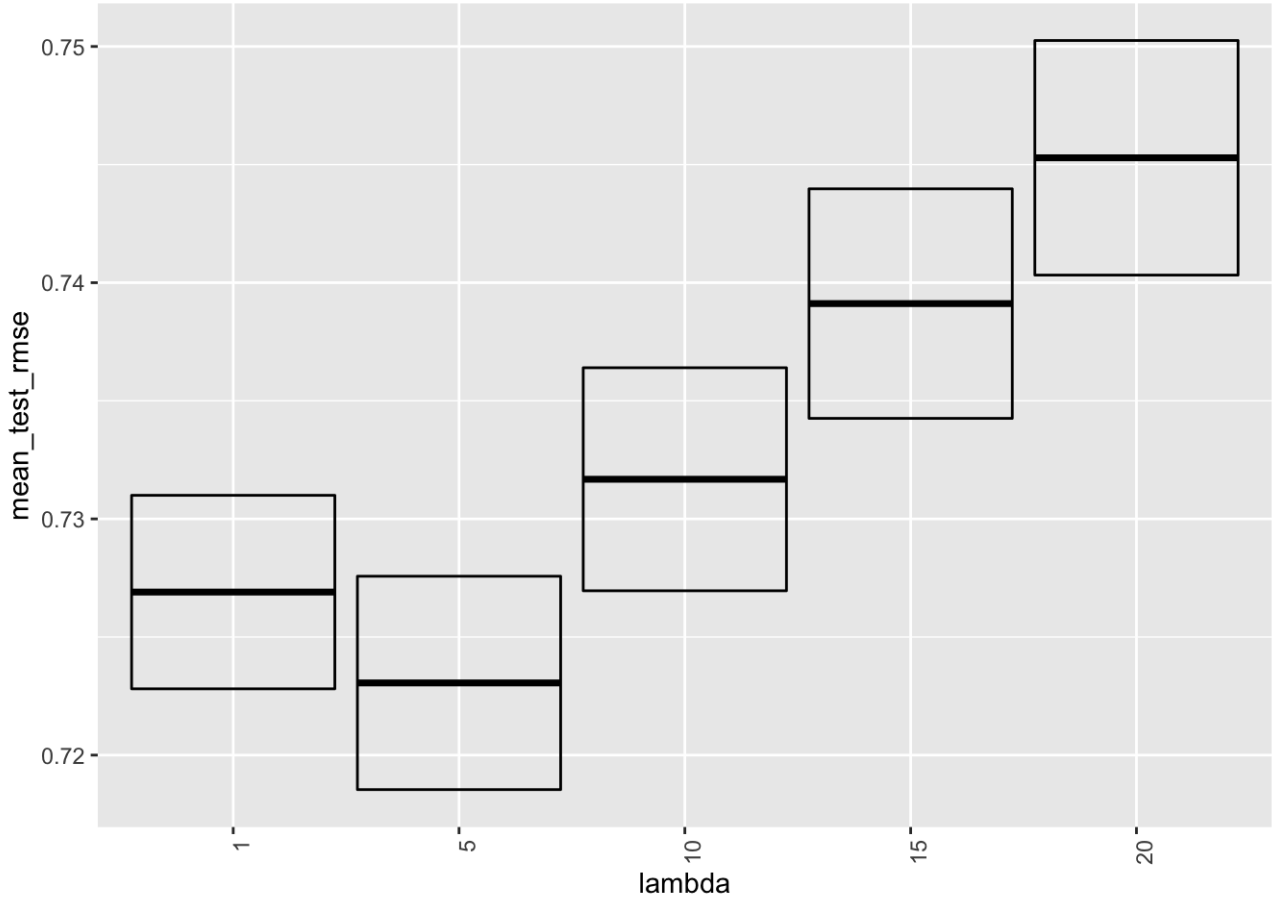
For further improvement, we can tune different Ks.

2. SVD with Kernel Ridge Regression

For Kernel Ridge Regression, we pick the Gaussian Kernel. We did the cross validation with lambda=1, 5, 10, 15, 20

```
source("../lib/KRR.Cross.Validation.R")
source("../lib/KRR.Postprocessing.R")
lambda=c(1, 5, 10, 15, 20)
krr.cv.rmse <- matrix(0, nrow = length(lambda), ncol = 4)
KRR.CV.Runtime <- system.time( for(i in 1:length(lambda)){
    cat("lambda=", lambda[i], "\n")
    krr.cv.rmse[i,] <- krr.cv (dat_train=data, K.fold=5, lambda=lambda[i])
    save(krr.cv.rmse, file="../output/krr.cv.rmse.RData")
  })
```

```
#Load visualization of cross validation results of KRR
load("../output/krr.cv.rmse.RData")
krr.cv.rmse <- as.data.frame(krr.cv.rmse)
colnames(krr.cv.rmse) <- c("mean_train_rmse", "mean_test_rmse", "sd_train_rmse", "sd_test
_rmse")
lambda=c(1,5,10,15,20)
krr.cv.rmse$lambda = as.factor(lambda)
krr.cv.rmse %>%
  ggplot(aes(x = lambda, y = mean_test_rmse,
             ymin = mean_test_rmse - sd_test_rmse, ymax = mean_test_rmse + sd_test_rmse))
+
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Visualize the cross validation testing error. We pick the best parameter lambda = 5.

```
source("../lib/KRR.Postprocessing.R")
KRR.result <- KRR.Post(lambda = 5, data=data, train = data_train, test=data_test)
save(KRR.result, file = "../output/KRR.result.RData")
```

```
## The train RMSE for SVD with Kernel Ridge Regression is 0.6099711 .
```

```
## The test RMSE SVD with Kernel Ridge Regression is 0.8608801 .
```

The result is close to the algorithm ALS.

Next, we fit the linear regression with the predictions from ALS algorithm and KRR.

```
data_train$krr.pred <- apply(data_train, 1, get.pred, est_rating=KRR.result$krr.rating)
data_test$krr.pred <- apply(data_test, 1, get.pred, est_rating=KRR.result$krr.rating)

## fit linear model
als.krr.model <- lm(rating ~ als.pred+krr.pred, data=data_train)

## get training prediction
als.krr.train <- predict(als.krr.model, data_train[,c(5,7)])


## get testing prediction
als.krr.test <- predict(als.krr.model, data_test[,c(5,7)])

## get training and testing RMSE
als.krr.train.rmse <- sqrt(mean((data_train$rating - als.krr.train)^2))
als.krr.test.rmse <- sqrt(mean((data_test$rating - als.krr.test)^2))
```

```
## The train RMSE for KRR after doing the linear regression with ALS is 0.5934735 .
```

```
## The test RMSE KRR after doing the linear regression with ALS is 0.877894 .
```

The testing RMSE increases a little bit. This might due to chances.

To further improve this process, we can tune more parameters. In addtion, we can also tune the KRR and ALS paramters at the same time. However, the process is going to be computational expensive.

Step 4 Evaluation

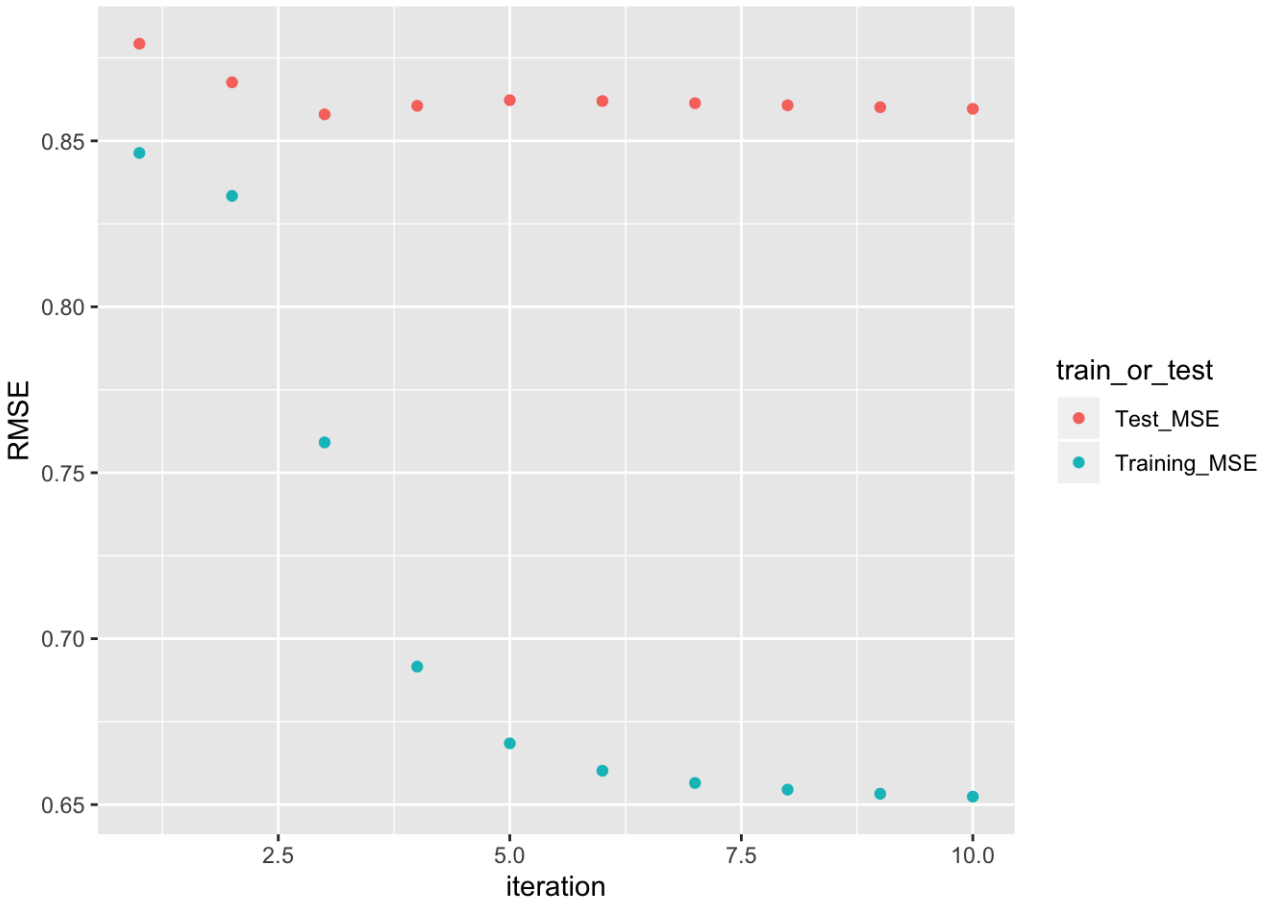
Visualization of training and testing RMSE by different dimension of factors and iterations

Best Parameters: lambda = 10 and f = 10. We do it with 10 iterations.

```
## run 10 iterations on the best parameters, and visualize the result
result.w.bp <- ALS.R1R2(f = 10, lambda = 10, max.iter=10, data=data, train=data_train, te
st=data_test)
save(result.w.bp, file = "../output/result.w.bp.RData")
```

```
load("../output/result.w.bp.RData")

RMSE <- data.frame(iteration = c(1:10), Training_MSE = result.w.bp$train_RMSE, Test_MSE =
result.w.bp$test_RMSE) %>% gather(key = train_or_test, value = RMSE, -iteration)
RMSE %>% ggplot(aes(x = iteration, y = RMSE,col = train_or_test)) + geom_point()
```



Note: For our final model, we used lambda = 10 and f = 10 with three iterations because the test RMSEs hardly change after three iterations. For the efficiency purpose, we choose max iteration=3.

Result Summary (RMSE result)

Show 10 entries Search:

	Training RMSE	Testing RMSE
ALS+R1+R2	0.7665	0.8576
KNN	1.1134	1.1485
KRR	0.61	0.8609
KNN+ALG Linear Regression	0.7462	0.8635
KRR+ALG Linear Regression	0.5935	0.8779

Showing 1 to 5 of 5 entries Previous 1 Next

Step 5 Bonus: New Evaluation Criteria

Since this is a movie recommendation system, we care about people's preference. Thus, we maily focus on high rating movies, and come up with a "High Rating Accuracy" measure

Assume: Rating >= 4 means people like the movie

To be conservative, we will recommend movies to users if the prediction rating is over 4.5

Formula: High rating movies accuracy = P(true rating >=4 | predict rating >=4.5)

Goal: Based on our prediction ratings, if we recommend movie to the users, how likely they will like them.

```
als.train.acc <- sum(subset(data_train,als.pred>=4.5)$rating>=4) / sum(data_train$als.pre
d>=4.5)
als.test.acc <- sum(subset(data_test,als.pred>=4.5)$rating>=4) / sum(data_test$als.pred>=
4.5)

knn.train.acc <- sum(subset(data_train,knn.pred>=4.5)$rating>=4) / sum(data_train$knn.pre
d>=4.5)
knn.test.acc <- sum(subset(data_test,knn.pred>=4.5)$rating>=4) / sum(data_test$knn.pred>=
4.5)

krr.train.acc <- sum(subset(data_train,krr.pred>=4.5)$rating>=4) / sum(data_train$krr.pre
d>=4.5)
krr.test.acc <- sum(subset(data_test,krr.pred>=4.5)$rating>=4) / sum(data_test$krr.pred>=
4.5)

data_train$knn.alg <- als.knn.train
data_test$knn.alg <- als.knn.test
data_train$krr.alg <- als.krr.train
data_test$krr.alg <- als.krr.test

knn.alg.train.acc <- sum(subset(data_train,knn.alg>=4.5)$rating>=4) / sum(data_train$knn.
alg>=4.5)
knn.alg.test.acc <- sum(subset(data_test,knn.alg>=4.5)$rating>=4) / sum(data_test$knn.alg
>=4.5)

krr.alg.train.acc <- sum(subset(data_train,krr.alg>=4.5)$rating>=4) / sum(data_train$krr.
alg>=4.5)
krr.alg.test.acc <- sum(subset(data_test,krr.alg>=4.5)$rating>=4) / sum(data_test$krr.alg
>=4.5)

acc.summary <- rbind (`ALS+R1+R2` = c(paste(round(als.train.acc*100,2),"%"), paste(round
(als.test.acc*100,2),"%")),
                      `KNN` = c(paste(round(knn.train.acc*100,2),"%"), paste(round(knn.te
st.acc*100,2),"%")),
                      `KRR` = c(paste(round(krr.train.acc*100,2),"%"), paste(round(krr.te
st.acc*100,2),"%")),
                      `KNN+ALG Linear Regression` =c(paste(round(knn.alg.train.acc*100,2
),"%"), paste(round(knn.alg.test.acc*100,2),"%")),
                      `KRR+ALG Linear Regression` =c(paste(round(krr.alg.train.acc*100,2
),"%"), paste(round(krr.alg.test.acc*100,2),"%")))

colnames(acc.summary) <- c("Training Accuracy", "Testing Accuracy")
datatable(acc.summary)
```

Show 10 entries Search:

	Training Accuracy	Testing Accuracy
ALS+R1+R2	96.87 %	93.88 %
KNN	62.56 %	54.17 %
KRR	99 %	94.35 %
KNN+ALG Linear Regression	93.5 %	90.88 %
KRR+ALG Linear Regression	98.44 %	90.48 %

Summary: our report shows that ALS with R1+R2 regularization has the lowest RMSE. ALS(with regularization)+KRR linear regression has the highest high rating accuracy. However, the results are close so they may due to chances. It is hard to tell which method is superior to the other one. What we only can tell that KNN post processing itself cannot improve the prediction.

Further improvement:

1. tune more parameters for post processing.
2. tune the parameters of Algorithem and post processing at the same time, instead of using the greedy method (what we are doing right now)
3. For checking the high rating accuracy, we can plot a ROC curve, like professor suggested in class, which can be more informative.