Final Prediction

Create edx and final_holdout_test sets

The code of this part was provided by professor, I just put it there to make sure all code could run.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Basic Overview

Before training a machine learning algorithm, I thought it necessary to make a quick data overview

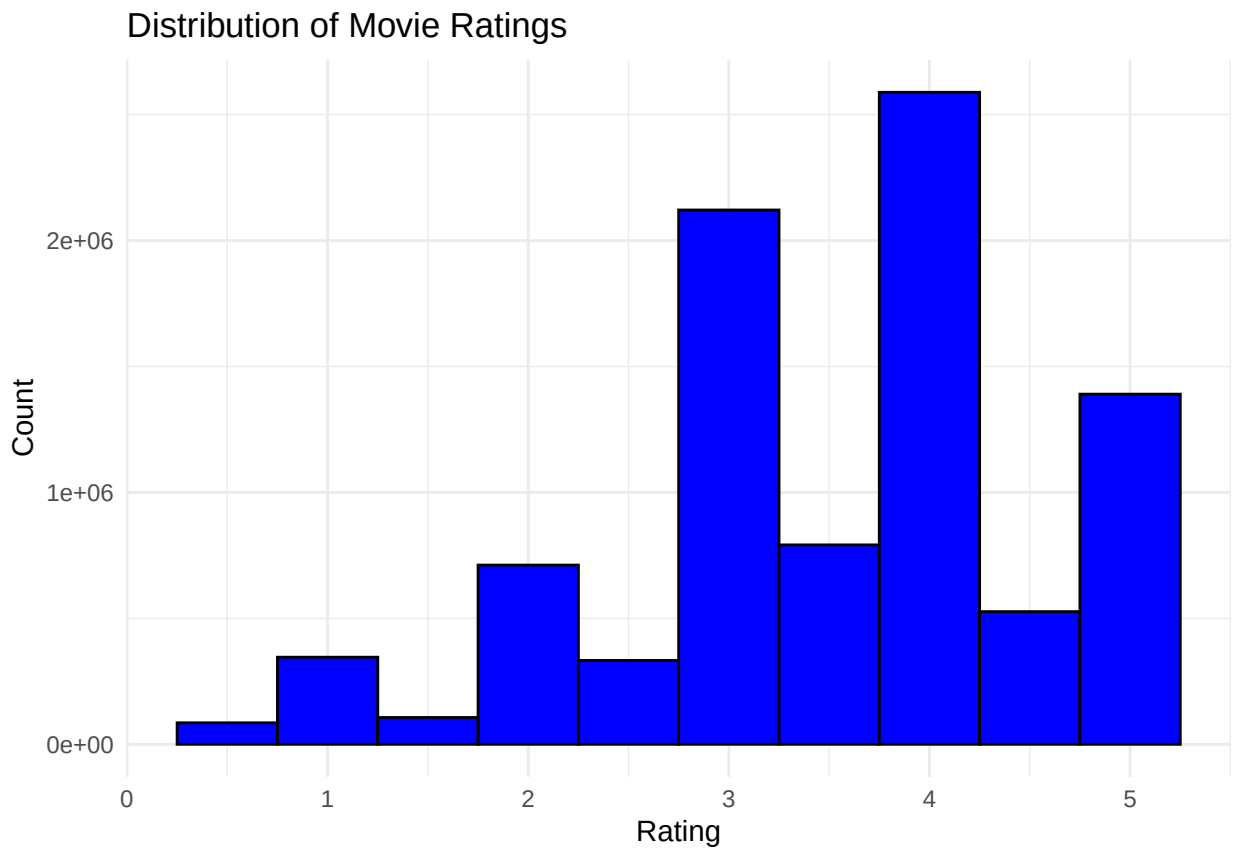
```
summary(edx) # Summary statistics
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
dim(edx) # Dimensions of the dataset
```

```
## [1] 9000055       6
```

```
# Distribution of Ratings
library(ggplot2)
ggplot(edx, aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "blue", color = "black") +
  theme_minimal() +
  labs(title = "Distribution of Movie Ratings", x = "Rating", y = "Count")
```




```
# Number of Ratings per Movie
edx %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  head(10) # Top 10 most rated movies
```

```
## # A tibble: 10 x 2
##    movieId count
##      <int> <int>
##  1     296 31362
##  2     356 31079
##  3     593 30382
##  4     480 29360
##  5     318 28015
##  6     110 26212
##  7     457 25998
##  8     589 25984
##  9     260 25672
## 10     150 24284
```

```
# Number of Ratings per User
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  head(10) # Top 10 most active users
```

```
## # A tibble: 10 x 2
##    userId count
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
```

Prediction of the Movie Ratings

- Random Forest

  Random forest is a classic machine learning algorithm, so I tried it first. But Then I found that the analysis is too complicated to do on my laptop. I can't using 'caret' package to build the model and check the validation. Since my device can't support me for complicated analysis, the random forest model I built is simple. Even though after over 8 hours calculation I got the result, Still having problems when I knit it to PDF. So in the Rmd file, I will show the code and results in this section. While in the pdf I will skip this part, just code and result in text.

```
# Random Forest #

library(randomForest)

library(Metrics)

# Splitting the edx dataset

set.seed(123)  # for reproducibility

ind <- sample(2, nrow(edx), replace = TRUE, prob = c(0.8, 0.2))

trainset <- edx[ind==1,]

testset <- edx[ind==2,]

# Adjust ntree and mtry as needed

# my device can't support me for complicated analysis

rfModel <- randomForest(rating ~ ., data = trainset, ntree = 5, mtry = 2)

predictions <- predict(rfModel, testset)

rmse_rf <- rmse(testset$rating, predictions)

rmse_rf_final <- rmse(final_holdout_test$rating, predictions)

print(paste("RMSE_rf:", rmse))

print(paste("RMSE_rf_final:", rmse_rf_final))

[1] "RMSE_rf: 1.05701342522051"
[1] "RMSE_rf_final: 1.16739250947879"
```

I'm not satisfied with the RMSE, so then I tried another method

- Collaborative Filtering Model

  Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It can help me to build a recommender system that can suggest movies that a user might enjoy, based on the ratings that the user and other users have given to different movies. On the other side, it can predict the ratings.

```
# Splitting the edx dataset into training and validation sets
set.seed(123)
train_indices <- sample(1:nrow(edx), 0.8 * nrow(edx))
train_set <- edx[train_indices, ]
validation_set <- edx[-train_indices, ]

# Implementing a Basic Collaborative Filtering Model #
# For this project, I use a simple mean-based approach
# If it can't fit, I will consider more sophisticated methods like matrix factorization

# Calculate mean ratings for each movie
movie_mean_ratings <- train_set %>%
  group_by(movieId) %>%
  summarise(mean_rating = mean(rating))

# Predicting ratings for the validation set
validation_set <- left_join(validation_set, movie_mean_ratings, by = "movieId")

# Handle movies not seen in the training set by assigning overall mean rating
overall_mean <- mean(train_set$rating, na.rm = TRUE)
validation_set$mean_rating[is.na(validation_set$mean_rating)] <- overall_mean

# This is a very basic approach
# RMSE Calculation
rmse <- sqrt(mean((validation_set$rating - validation_set$mean_rating)^2))
print(paste("Validation RMSE:", rmse))
```

```
## [1] "Validation RMSE: 0.943715211794902"
```

```
# Final Evaluation on final_holdout_test set
# Similar steps as validation, but using the final_holdout_test dataset

final_holdout_test <- left_join(final_holdout_test, movie_mean_ratings, by = "movieId")
final_holdout_test$mean_rating[is.na(final_holdout_test$mean_rating)] <- overall_mean

final_rmse <- sqrt(mean((final_holdout_test$rating - final_holdout_test$mean_rating)^2))
print(paste("Final RMSE:", final_rmse))
```

```
## [1] "Final RMSE: 0.944090516186475"
```

Since Collaborative Filtering algorithm is a quite basic approach, although the RMSE is satisfying, I thought it would be better trying some other algorithms. So I tried SVD(single value decomposition) to compare with collaborative filtering model and random forest.

- SVD(single value decomposition)

  SVD stands for Singular Value Decomposition, which is a way of breaking down a matrix into three smaller matrices.The SVD can be used for many purposes, such as finding the best approximation of A with a lower rank, finding the pseudo-inverse of A, or finding the rank, range, and null space of A.

```
library(data.table)
library(recommenderlab)
library(Metrics)

# Convert the edx dataset to a data.table
edx_dt <- as.data.table(edx)

# Create a sparse matrix using recommenderlab
rating_matrix <- as(edx_dt, "realRatingMatrix")

# Splitting the dataset in another way
train_test_split <- sample(c(TRUE, FALSE), nrow(rating_matrix), replace = TRUE, prob = c(0.8, 0.2))
train_matrix <- rating_matrix[train_test_split]
test_matrix <- rating_matrix[!train_test_split]
# Train the SVD model
svd_model <- Recommender(train_matrix, method = "SVD", parameter = list(k = 20))
# Making predictions
predicted_ratings <- predict(svd_model, test_matrix, type = "ratings")

# Convert predicted ratings to a regular matrix
predicted_matrix <- as(predicted_ratings, "matrix")
actual_matrix <- as(test_matrix, "matrix")

# Replace NAs in predicted_matrix with the mean rating (or another appropriate value)
mean_rating <- mean(actual_matrix, na.rm = TRUE)
predicted_matrix[is.na(predicted_matrix)] <- mean_rating

# Extracting the overlapping entries (non-NA in both actual and predicted matrices)
overlap <- !is.na(actual_matrix) & !is.na(predicted_matrix)
#rmse <- function(actual, predicted) {
#  sqrt(mean((actual - predicted)^2))
#}
# Calculate RMSE only on overlapping entries
rmse_SVD <- sqrt(mean((actual_matrix[overlap] - predicted_matrix[overlap])^2))
print(paste("RMSE with SVD Model:", rmse_SVD))
```

```
## [1] "RMSE with SVD Model: 1.05701342522051"
```

The RMSE with SVD Model is even bigger than random forest and basic collaborative filtering model, so I skip using the final_holdout_test set to calculate the RMSE.

Conclusion

Comparing the RMSE of the 3 algorithms, Basic Collaborative Filtering Model seemed to be the best one with RMSE less than 1. The final RMSE is 0.9441.

```
print(paste("Final RMSE:", final_rmse))
```

```
## [1] "Final RMSE: 0.944090516186475"
```