HarvardX Data Science Part 9: Capstone (Movie Recommendation System)

Ma Rui Chen, Ryan

4/9/2020

Overview

- Using the edx dataset generated from the code provided, we have constructed a model that can predict what rating a user would give a movie with a root-mean-square error of around 0.865.
- Our model is as follows: Rating given by user u for movie i = mu + b_i + b_u + epsilon, where mu is the mean rating in the edx dataset, b_i is the effect of each different movie, b_u is the effect of each different user, and epsilon is the error term with mean 0.
- In our model, b_i and b_u are regularised.
- Our model is imperfect. After removing the movie and user effects, the error term still contains the effect caused by movies grouped by genre, series, etc. We propose using the genres as predictors to train logistic regression (GLM), linear and quadratic discrimination analyses (LDA and QDA), k-nearest-neighbors (kNN), and random forest algorithms. We would then choose the algorithm that gives the lowest RMSE. Regrettably, we lack the computational power to follow through with this method, as fitting these models caused our laptop to hang.

Pre-Requisite: Generate training (edx) and validation Datasets

```r
library(caret)
library(data.table)
library(stringr)
library(tidyverse)

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE) # Validation
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId") # Make sure userId and movieId in validation set are also in edx set

removed <- anti_join(temp, validation) # Add rows removed from validation set back into edx set

edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed) #remove these objects from environment
```

Methods

In our recommendation system, the outcome we're trying to predict is the rating, and each of the other column headings (userID, movieID, timestamp, and genres) are potential predictors. We first choose our predictors by examining their variability in the edx dataset.

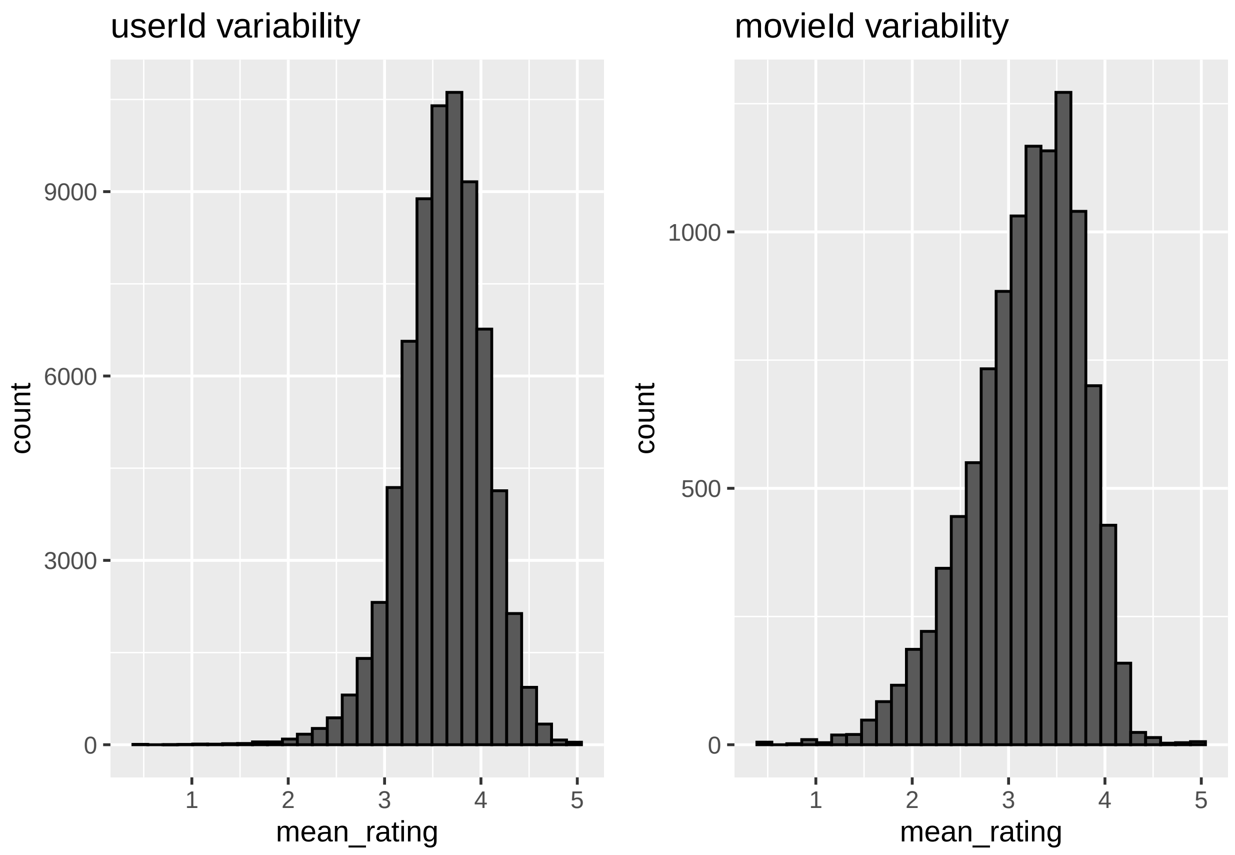
Below are histograms illustrating the variabilities of mean ratings grouped by userId and movieId.

```
library(tidyverse)

#userId
p1 <- edx %>%
  group_by(userId) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins=30, col=I("black")) +
  ggtitle("userId variability")

#movieId
p2 <- edx %>%
  group_by(movieId) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins=30, col=I("black")) +
  ggtitle("movieId variability")

library(gridExtra)
par(mfrow=c(1,2))
grid.arrange(p1, p2, nrow=1)
```



As seen, there is significant variability in the ratings when grouped by movieId and userId, hence these predictors should be accounted for in our model.

Baseline RMSE

We first determine what RMSE we would get if we used the simplest model, i.e. predicting all ratings in the validation set to be the mean rating in the edx set. To do so, we first define a function to calculate RMSE:

```
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

We then calculate the RMSE of our first model and save it in a data frame for comparison purposes later on:

```
mu <- mean(edx$rating)
y_hat <- rep(mu, nrow(validation))
model_1_rmse <- RMSE(validation$rating, y_hat)

rmse_results <- tibble(Method = "Just the mean",
                RMSE = model_1_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   Method         RMSE
##   <chr>          <dbl>
## 1 Just the mean  1.06
```

Movie Effect

Our current model is as follows:

Yi = mu + b_i + b_u + epsilon, where Yi is the predicted rating for movie i, mu is the mean rating in the training set, b_i is the effect of each different movie, b_u is the effect of each different user, and epsilon is the error term with mean 0. We first find b_i for each movie:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating-mu))
head(movie_avgs)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.415
## 2       2 -0.307
## 3       3 -0.365
## 4       4 -0.648
## 5       5 -0.444
## 6       6  0.303
```

User Effect

Next, we find b_u for each user:

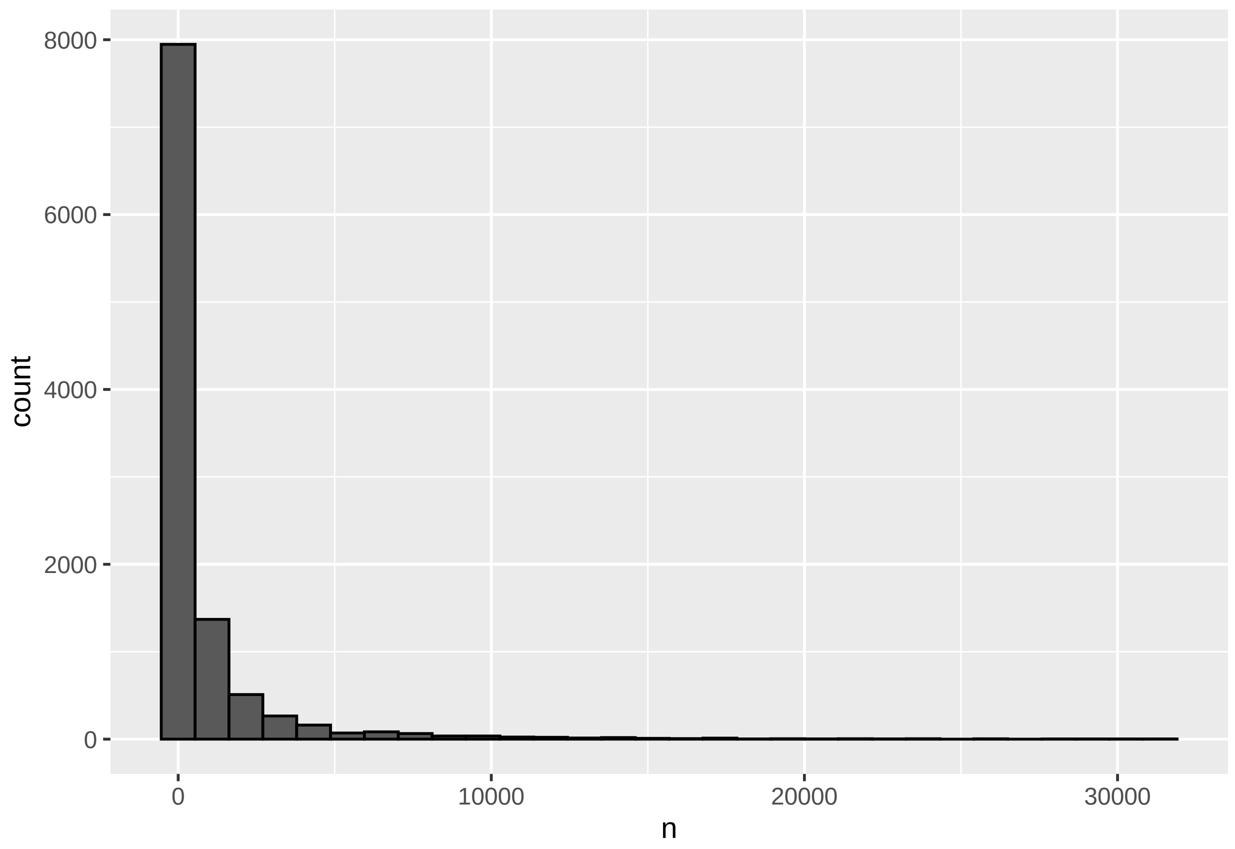
```
user_avg <- edx %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))
head(user_avg)
```

```
## # A tibble: 6 x 2
##   userId     b_u
##    <int>   <dbl>
## 1      1  1.68
## 2      2 -0.236
## 3      3  0.264
## 4      4  0.652
## 5      5  0.0853
## 6      6  0.346
```

Regularisation

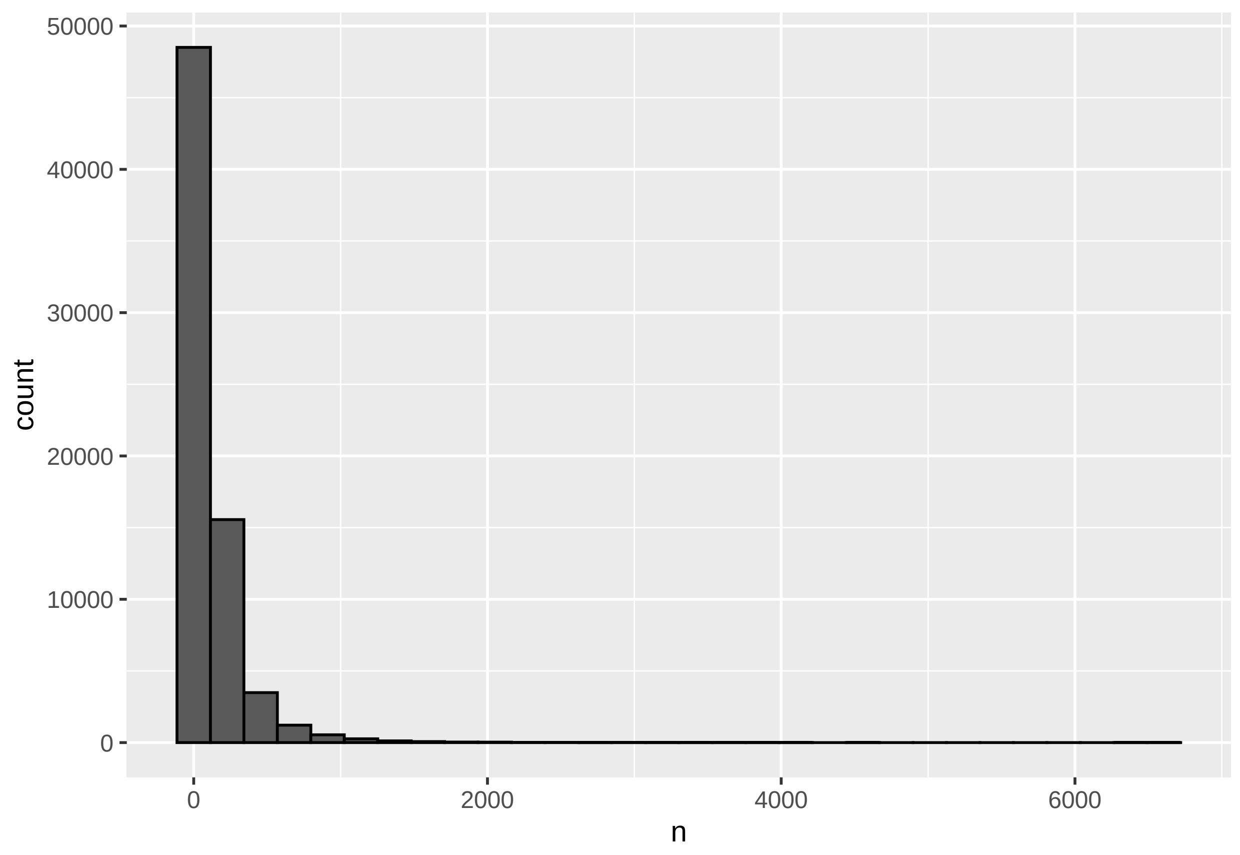
As seen in the histogram below, not all movies have a large number of ratings; the vast majority have very few.

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(col=I("black"))
```



Not all users have rated many movies either, as seen below:

```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(col=I("black"))
```



For movies and users with few ratings, extreme values will be more common, making the rating recorded less reliable. Hence, we need to regularise the ratings, in order to reduce the weightage of movies and users with few ratings.

To regularise the ratings, we will divide the sum of the effect in question by (n + lambda), where n is the number of counts of the movie or user, and lambda is a tuning parameter which we can find via cross-validation.

```
library(caret)

lambdas <- seq(0, 10, 0.25)
reg_rmses <- sapply(lambdas, function(l) {

  test_index <- createDataPartition(edx$rating, times=1, p=0.1, list=F)
  train_set <- edx[-test_index,]
  temp <- edx[test_index,]
  cv_set <- temp %>%
    semi_join(train_set, by="movieId") %>%
    semi_join(train_set, by="userId")
  removed <- anti_join(temp, cv_set)
  train_set <- rbind(train_set, removed)

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating-mu)/(l+n()))
```

```r
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating-mu-b_i)/(l+n()))

  predicted_ratings <- cv_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(predicted_ratings, cv_set$rating))
})

# Most optimal lambda
lambdas[which.min(reg_rmses)]
```

```
## [1] 1.5
```

As seen, the most optimal lambda to use is 1.5. Now we will regularise the movie and user effects.

```r
l <- 1.5

# Movie effect
b_i <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating-mu)/(l+n()))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating-mu-b_i)/(l+n()))
```

Now let's review our RMSE based on our current model of regularised movie and user effects.

```r
y_hat <- validation %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_2_rmse <- RMSE(validation$rating, y_hat)
rmse_results <- rbind(rmse_results, data.frame(Method = "Regularised Movie + User Effects",
                                               RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   Method                            RMSE
##   <chr>                            <dbl>
## 1 Just the mean                    1.06
## 2 Regularised Movie + User Effects 0.865
```

As seen, our RMSE has improved to about 0.865.

In the next section, we will present our unfinished work - training algorithms to account for the effect of movie genres. Unfortunately, we lack the computing power to finish running the code.

Genre Effect (Unfinished)

As we have seen in the course using the smaller dslabs movielens dataset, the movie and user effects still do not fully explain the variation in ratings. Within certain groups of movies, ratings are still very much correlated. These groups can be based on genre, series, etc.

To illustrate this, we take a small subset of the edx dataset, and plot the relationship between the residual ratings of two movies in the Terminator series after removing the movie and user effects.

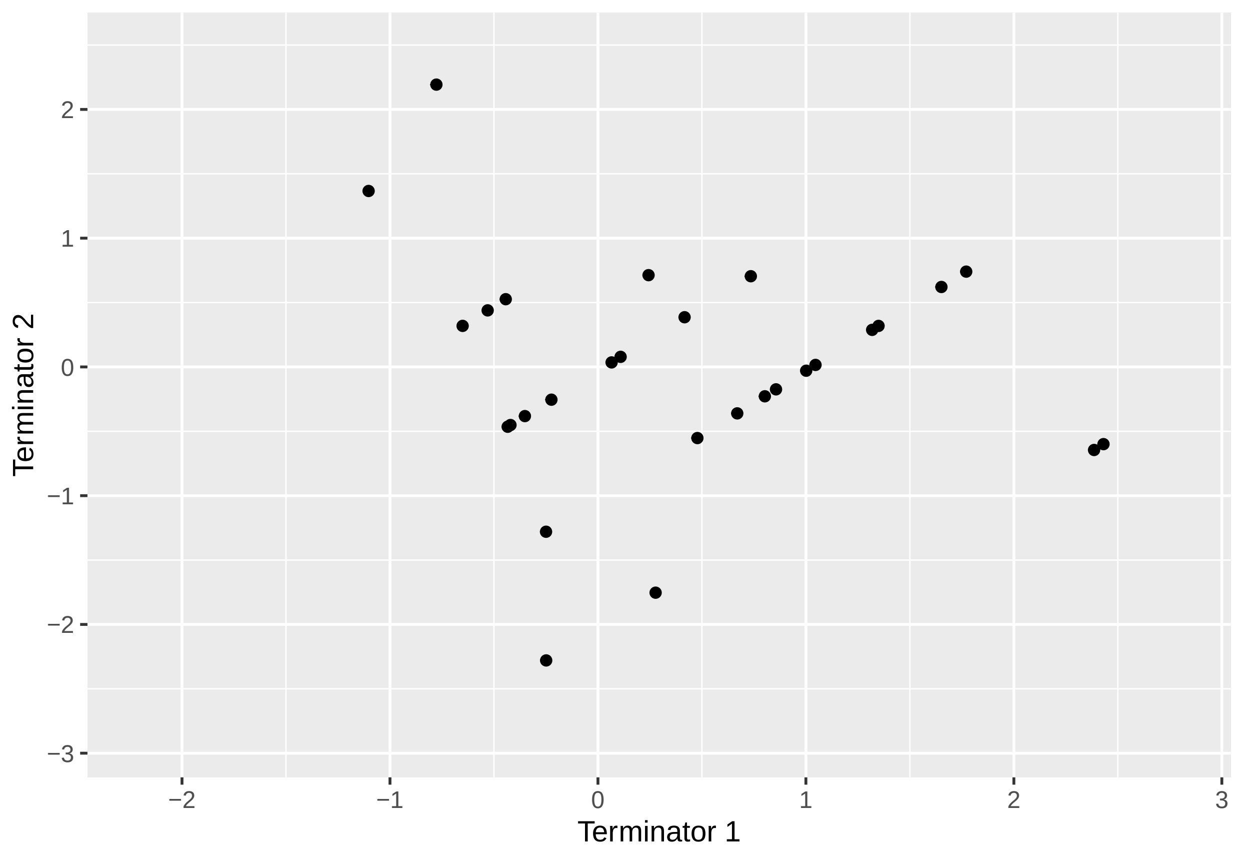
```r
foo <- edx %>%
  group_by(movieId) %>%
  filter(n() >= 50) %>%
  ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 50) %>%
  ungroup() %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix

foo <- foo[,-1]

foo <- sweep(foo, 1, rowMeans(foo, na.rm=T))
foo <- sweep(foo, 2, colMeans(foo, na.rm=T))

qplot(foo[, 1240], foo[, 589],
      xlab = "Terminator 1",
      ylab = "Terminator 2")
```

```
## Warning: Removed 40566 rows containing missing values (geom_point).
```



As seen, the residual ratings of the two Terminator movies are highly correlated when they should be independent. Hence, our model must taken into account group effect. To do so, we train the residual rating using logistic regression, k-nearest neighbors, linear and quadratic discriminant analysis, and random forest.

First we create a dataset with the residual ratings after removing the movie and user effects. We then split this dataset into training and cross-validation sets.

```
edx_resid <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  mutate(resid = rating - mu - b_i - b_u)

cv_ind <- createDataPartition(edx_resid$resid, times=1, p=0.2, list=F)
resid_train <- edx_resid %>% slice(-cv_ind)
resid_cv <- edx_resid %>% slice(cv_ind)
```

Next, we separate the genres column into multiple columns, each containing one genre.

```
genres <- separate(data = edx_resid,
         col = genres,
         into = paste0("genre", 1:8), # 8 is the maximum number of genres a movie has
         sep = "\\|",
         remove = T) %>%
  select(6:13) %>%
  as.matrix
```

```r
#Convert NAs to 0
genres[is.na(genres)] <- 0

#Split genres into train and cv
genres_train <- genres[-cv_ind,]
genres_cv <- genres[cv_ind,]
```

We now train the genres matrix using logistic regression, LDA, and QDA.

```r
# Logistic regression
fit_glm <- train(genres_train, resid_train$resid,
                 method="glm")
cv_glm <- predict(fit_glm, genres_cv)

### Section incomplete as above code runs indefinitely
```

Results

As shown in an earlier section, our model yields an RMSE of about 0.865 on the validation set, which is short of the RMSE of 0.8567 achieved by the winners of the Netflix Prize in 2009.

```r
rmse_results
```

```
## # A tibble: 2 x 2
##   Method                            RMSE
##   <chr>                            <dbl>
## 1 Just the mean                    1.06
## 2 Regularised Movie + User Effects 0.865
```

Conclusion - What Else Could Have Been Done?

Firstly, with more computing power, we would have been able to train the GLM, LDA, QDA, kNN, and random forest algorithms as planned, and yielded a lower RMSE by accounting for the effect of movie genres.

Secondly, with more information, the timestamp column in the edx dataset could be used as a predictor as well, as people's opinions of movies sometimes change when a long time passes after these movies are released. This is especially relevant as some movies in the edx dataset were released as early as 1995, as seen below:

```r
library(lubridate)
edx %>%
  .$timestamp %>%
  as_datetime() %>%
  min
```

```
## [1] "1995-01-09 11:46:49 UTC"
```

If we had access to the release date of each movie in the data, we could have calculated the time elapsed since each movie's release date. We could then train an algorithm to account for the time effect.

Lastly, as with most recommendation systems, the accuracy of our model depends heavily on recording each user's activities, which would form their b_u predictor. If a new user starts using the platform, then our recommendation for them would only be based on mu + b_i until we get a more extensive record of their movie-watching habits. Of course, if we have access to new users' personal information, e.g. age, gender, and location, then these could be used to estimate their b_u based on other users in their demographic group.