Philipp Movielens Project

Nico Philipp

2/29/2024

Movie Lens Project Introduction

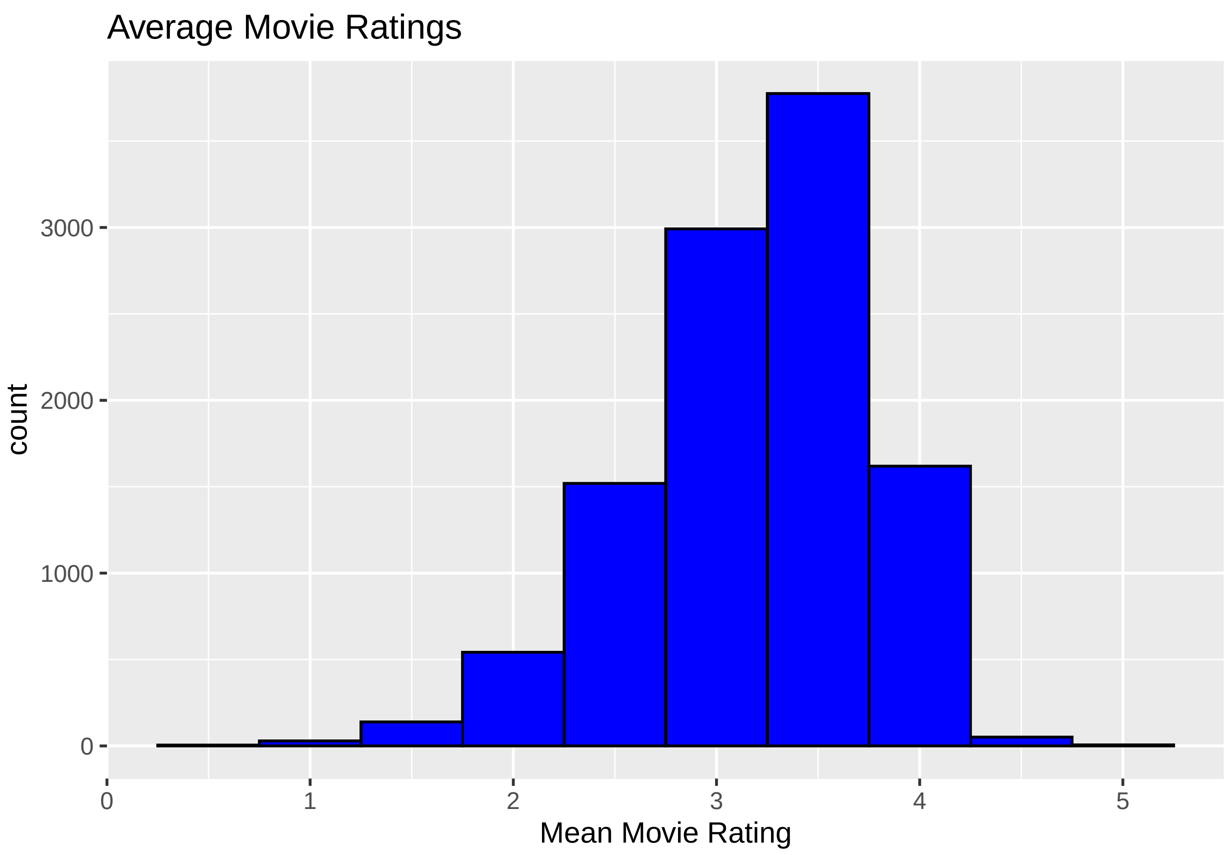
The overarching aim of this machine learning-based project was to use provided data from the movielens data set to develop a recommendation system for movies. To do so we use our edx data set, and break it into a training and a testing set to develop the models. Finally, to check our models on new, unseen data, we introduce the final_holdout_test data set to make prediction on the validation data set. To compare and select models, we use root mean square error (rmse) as a selection tool. In summary, we are trying to predict the rating given by a certain user.

The following are features we may use to make predictions: *userID*, *movieID*, *timestamp*, *genre*, *title*

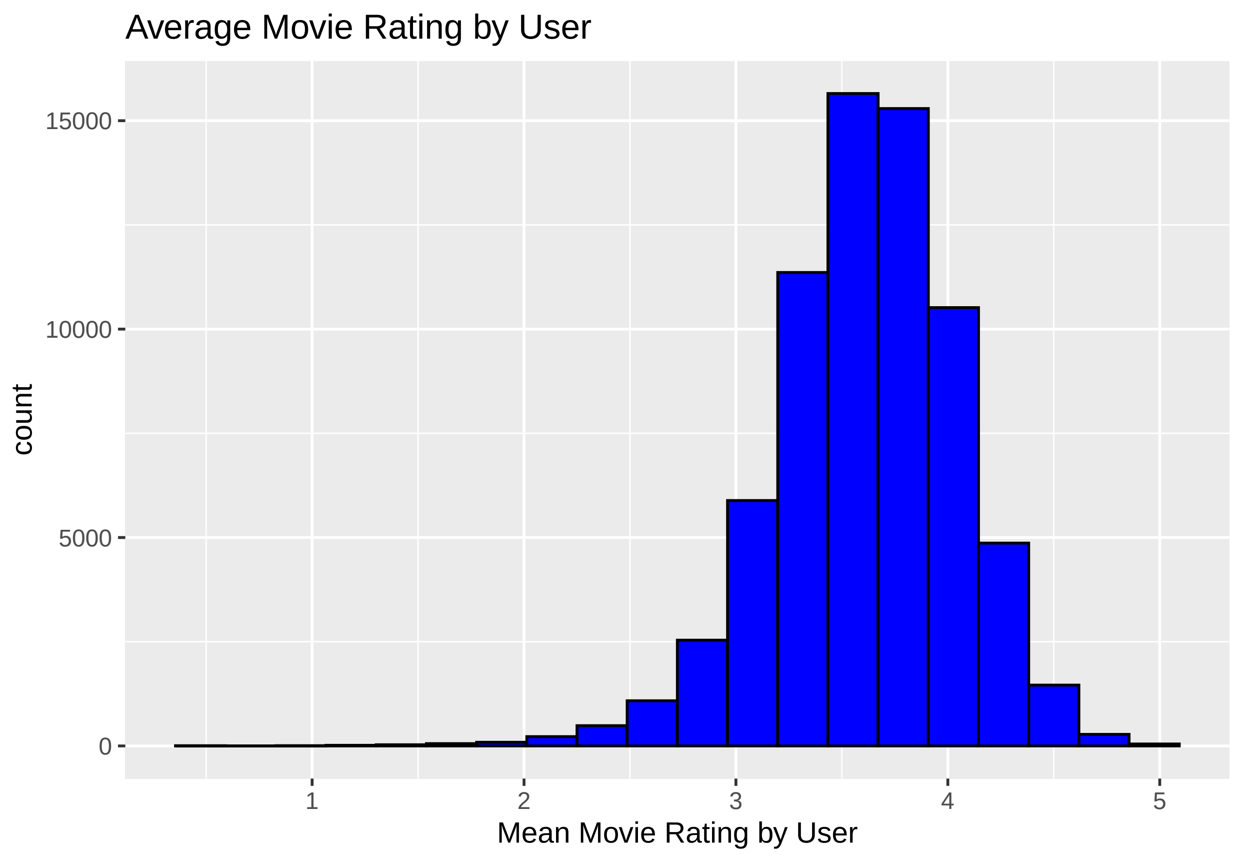
```r
#Download data and load libraries
suppressMessages({

download.file("https://www.dropbox.com/s/4t4w48ut2gu8dn6/edx.rds?dl=1", "edx.rds")

download.file("https://www.dropbox.com/s/c4jzznttgc01sdb/final_holdout_test.rds?dl=1", "final_holdout_te

edx = readRDS("edx.rds")

final_holdout_test = readRDS("final_holdout_test.rds")

str(edx)

### Load libraries
library(tidyverse)
library(caret)
})
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
```

Plotting Data

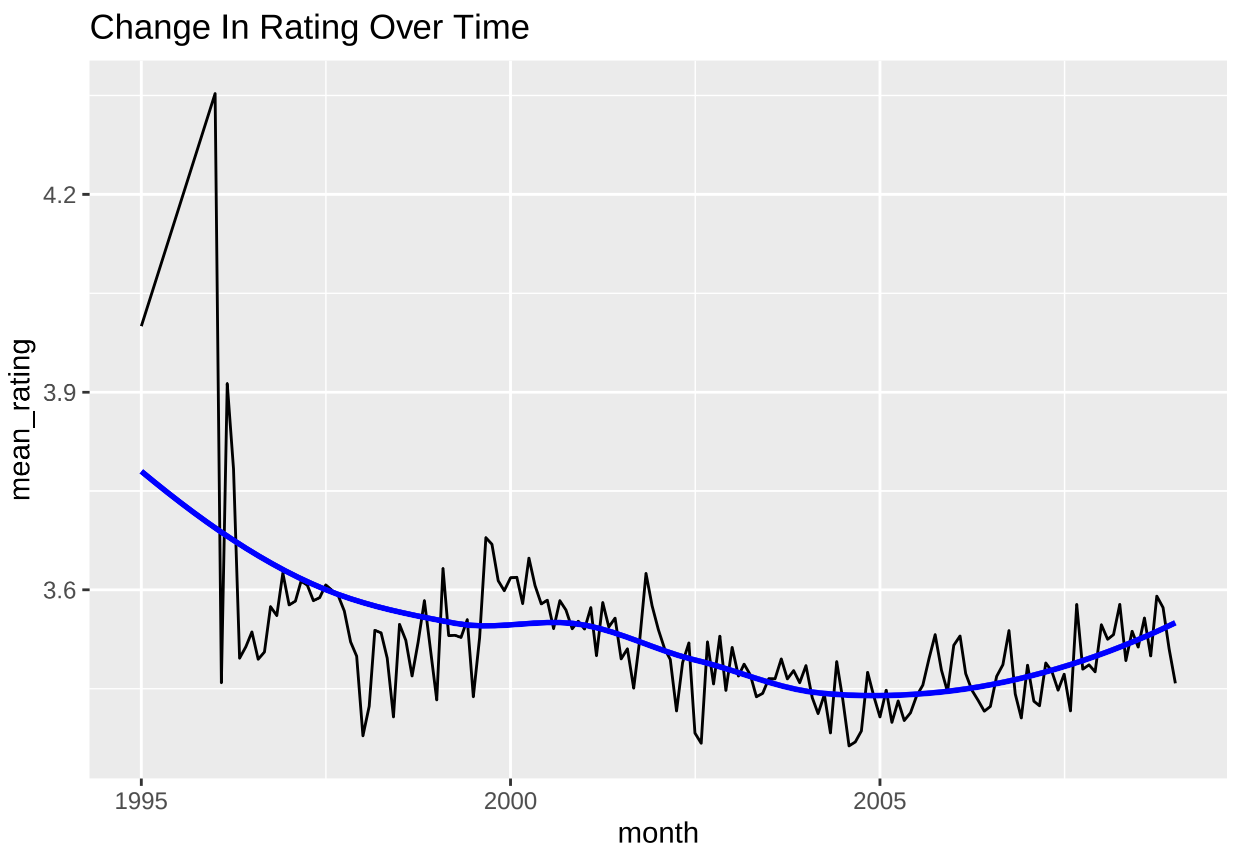


We identify there is considerable variability amongst movie ratings



We identify there to be considerable variability amongst users as well, which may be something we want to account for in developing our models

```
## `geom_smooth()` using formula = 'y ~ x'
```



The change in rating over time plot shows us some abnormally high ratings shortly after 1995.Therefore, we might consider only taking data from after 1998.

```
# Filter for timepoints after 1998
suppressMessages({

edx <- edx %>%
  filter(year(as_datetime(timestamp)) > 1998)

###apply the same to the holdout set
final_holdout_test <- final_holdout_test %>%
  filter(year(as_datetime(timestamp)) > 1998)
})
```

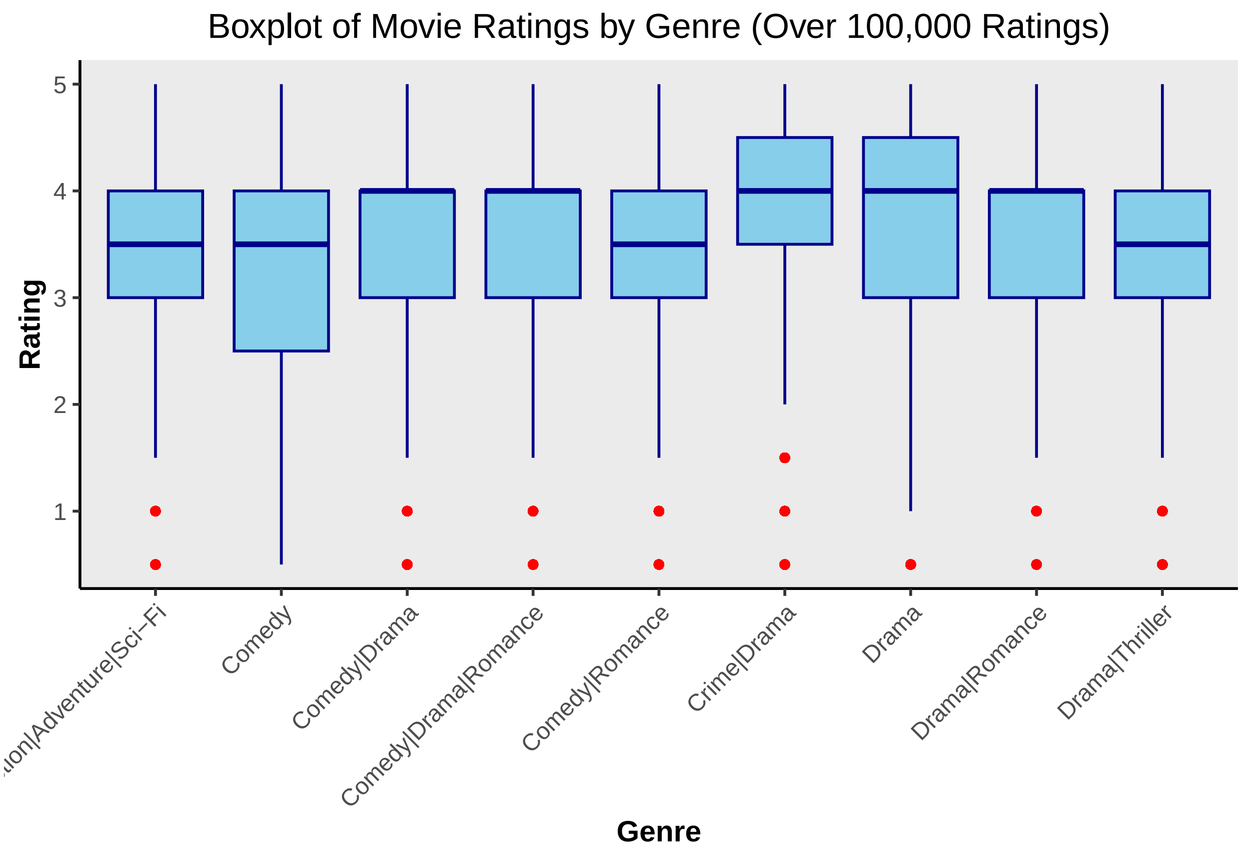
Knowing there are quite a few genres, we want to find out how many there are exactly.

```
## Number of unique genres: 797
```

797 is quite a few genres to plot. Plot only genres with a certain amount of ratings (i.e., >100000 rating)

```
# Calculate average rating and count of ratings for each genre
suppressMessages({

genre_stats <- edx %>%
  group_by(genres) %>%
  summarise(mean_rating = mean(rating), rating_count = n())

# Filter genres with over 100,000 ratings
filtered_genres <- genre_stats %>%
  filter(rating_count > 100000)

# Filter original data frame based on selected genres
edx <- edx %>%
  filter(genres %in% filtered_genres$genres)

#apply the same to the validation dataset
final_holdout_test <- final_holdout_test %>%
  filter(genres %in% filtered_genres$genres)
})
```

Create a boxplot for genres



Interestingly, comedy by itself while having a very wide range of rating, also has the lowest median rating. On the other hand, when combined with drama, it has the highest median rating. Drama and War also has a surprisingly high average rating.

Start of Model Generation

We apply an 80:20 train to test split. This split strikes a balance between having enough data for training to capture the underlying patterns in the data and having enough data for testing to evaluate the model's performance effectively. This split may further be adjusted based on model performance. For instance, if our model is overfitting, we may try a 70:30 split.

```
###setting the seed
suppressMessages({

set.seed(123)
###generate index for splitting the data (80 to 20 split)
splitIndex <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)

#Create training and testing datasets
train_data <- edx[-splitIndex, ]
test_data <- edx[splitIndex, ]

#ensure the userID and movieOD in the test set are also in the train set
test_data <- test_data %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")
})
```

Build a first simple recommendation model using naive bayes

```
suppressMessages({

mu <- mean(train_data$rating)
mu

naive_rmse <- RMSE(test_data$rating, mu)
naive_rmse

###store results in a dataframe

rmse_results_df <- data.frame(Model = "Mean Only", RMSE = naive_rmse)
print.data.frame(rmse_results_df)
})
```

```
##       Model    RMSE
## 1 Mean Only 1.05191
```

Next we add movie effects to our model to see how this affects our rmse

```
# Estimate movie effect (b_i)
suppressMessages({

b_i <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

###
predicted_ratings <- mu + test_data %>%
  left_join(b_i, by="movieId") %>%
  .$b_i

###
model_1_rmse <- RMSE(predicted_ratings, test_data$rating)

###store in data frame

rmse_results_df <- bind_rows(rmse_results_df,
                             data.frame(Model = "Movie Effect",
                                        RMSE = model_1_rmse))
rmse_results_df

})
```

```
##          Model      RMSE
## 1    Mean Only 1.0519097
## 2 Movie Effect 0.9413492
```

We see that adding movie effects decreased our rmse value

Next we add user effects to our model

```
# Estimate user effect
suppressMessages({

user_avgs <- train_data %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

###
predicted_ratings <- test_data %>%
  left_join(b_i, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)


model_2_rmse <- RMSE(predicted_ratings, test_data$rating)
model_2_rmse

###add to data frame

rmse_results_df <- bind_rows(rmse_results_df,
                             data.frame(Model = "Movie + User Effect",
                                        RMSE = model_2_rmse))
rmse_results_df

})
```

```
##                 Model      RMSE
## 1           Mean Only 1.0519097
## 2        Movie Effect 0.9413492
## 3 Movie + User Effect 0.8669997
```

We see that our rmse value further decreases, indicating model improvement

---

Next we add time effects to our model

```
# Estimate time effect
suppressMessages({

b_t_train <- train_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(time = as_datetime(timestamp)) %>%
  group_by(day = floor_date(time, "day")) %>%
  summarise(b_t = sum(rating - mu - b_i - b_u)/n())

# Predict ratings with movie, user, and time effects
predicted_ratings <- test_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
  left_join(b_t_train, by = "day") %>%
  mutate(b_t = replace_na(b_t, 0)) %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  pull(pred)

# Calculate RMSE for the model with movie, user, and time effects
model_3_rmse <- RMSE(predicted_ratings, test_data$rating)

# Add to the results dataframe
rmse_results_df <- bind_rows(rmse_results_df,
                             data.frame(Model = "Movie + User + Time Effect",
                                        RMSE = model_3_rmse))
print.data.frame(rmse_results_df)

})
```

```
##                        Model      RMSE
## 1                  Mean Only 1.0519097
## 2               Movie Effect 0.9413492
## 3        Movie + User Effect 0.8669997
## 4 Movie + User + Time Effect 0.8665766
```

Our rmse further decreases with the addition of time effects

Add genre effects

```
# Estimate genres effect
suppressMessages({

b_g_train <- train_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - mu - b_i - b_u)/n())

# Predict ratings with movie, user, and genres effects
predicted_ratings <- test_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(b_g_train, by = "genres") %>%
  mutate(b_g = replace_na(b_g, 0)) %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
```

```r
# Calculate RMSE for the model with movie, user, and genres effects
model_4_rmse <- RMSE(predicted_ratings, test_data$rating)

# Add to the results dataframe
rmse_results_df <- bind_rows(rmse_results_df,
                             data.frame(Model = "Movie + User + Genres Effect",
                                        RMSE = model_4_rmse))
print.data.frame(rmse_results_df)

})
```

```
##                          Model      RMSE
## 1                    Mean Only 1.0519097
## 2                 Movie Effect 0.9413492
## 3          Movie + User Effect 0.8669997
## 4   Movie + User + Time Effect 0.8665766
## 5 Movie + User + Genres Effect 0.8669257
```

Our rmse value does not further decrease through an addition of genre effects. However, there may be something we can do about this. The bar plot from earlier depicts that some genres have rather large standard errors. This is something that we may want to account for by controlling for high standard errors.

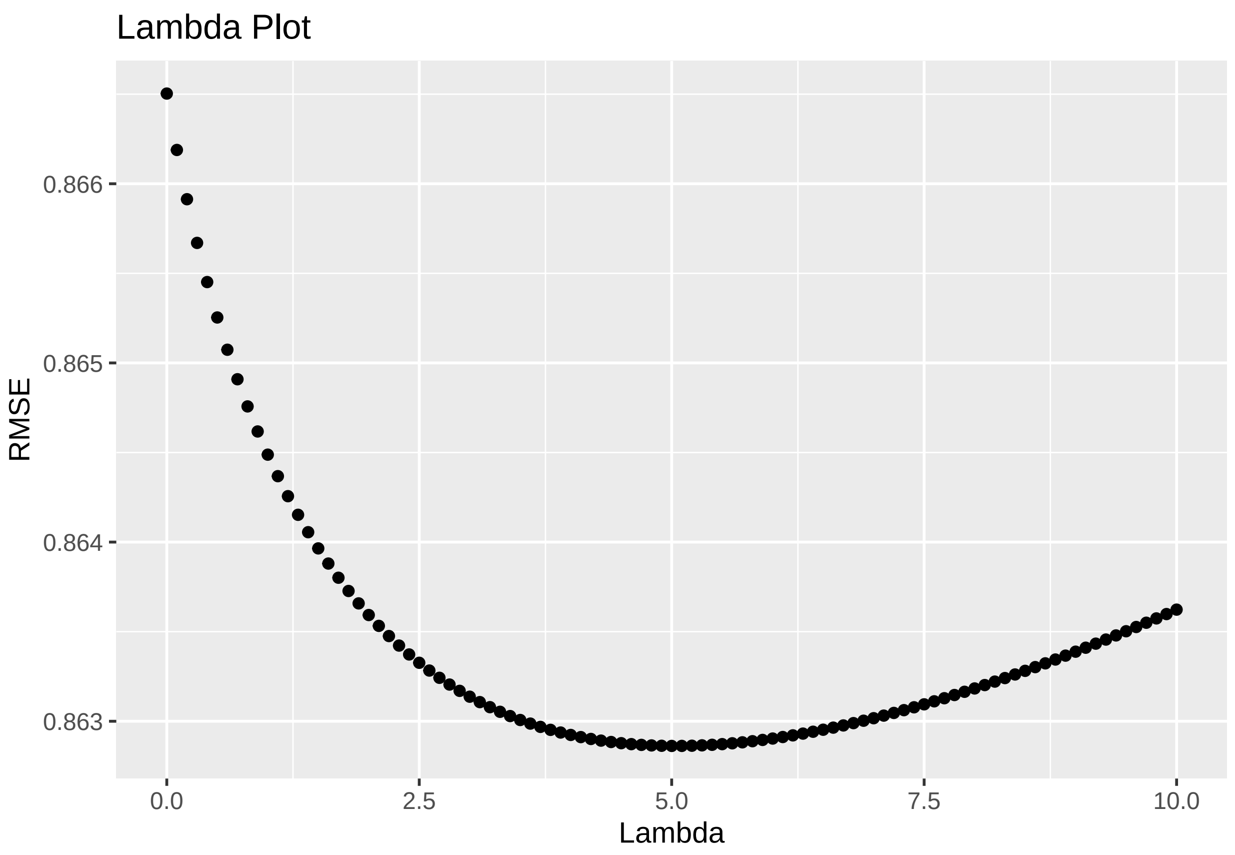
Perform Cross Validation

```r
suppressMessages({

# Specify standard error thresholds
# Specify the range of standard error thresholds
ses <- seq(0, 1, 0.1)

rmse <- sapply(ses, function(s){
  b_g <- train_data %>%
    left_join(user_avgs, by="userId") %>%
    left_join(b_i, by = "movieId") %>%
    mutate(day = floor_date(as_datetime(timestamp),"day")) %>%
    left_join(b_t_train, by="day") %>%
    mutate(b_t=replace_na(b_t,0)) %>%
    group_by(genres) %>%
    summarise(b_g=(sum(rating-b_i-b_u-b_t-mu))/(n()),se = sd(rating)/sqrt(n())) %>%
    filter(se<=s) # Retaining b_g values that correspond to Standard Error less than or equal to S

  # Predicting movie ratings on test set
  predicted_ratings <- test_data %>%
    left_join(b_i, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    mutate(day = floor_date(as_datetime(timestamp),"day")) %>%
    left_join(b_t_train, by="day") %>%
    mutate(b_t=replace_na(b_t,0)) %>%
    left_join(b_g, by="genres") %>%
    mutate(b_g=replace_na(b_g,0)) %>%
    mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
    .$pred
  RMSE(predicted_ratings, test_data$rating)
  return(RMSE(predicted_ratings, test_data$rating))
})

#Identify and store the lowest standard error
s_e <- ses[which.min(rmse)]
s_e

})
```

```
## [1] 0.1
```

Perform regularization to further improve results

```
suppressMessages({

 #Specify lambda values
lambdas <- seq(0, 10, 0.1)

 #Initialize an empty dataframe to store results
regularization_results_df <- data.frame(Lambda = numeric(),
                                        RMSE = numeric(),
                                        stringsAsFactors = FALSE)

# Iterate over lambda values
for (l in lambdas) {
  mu <- mean(train_data$rating)

   #Estimate movie effect (b_i)
  b_i <- train_data %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + l))

  # Estimate user effect (b_u)
  b_u <- train_data %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + l))

  # Estimate time effect (b_t)
  b_t <- train_data %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_i, by = "movieId") %>%
    mutate(time = as_datetime(timestamp)) %>%
    group_by(day = floor_date(time, "day")) %>%
    summarize(b_t = sum(rating - b_i - b_u - mu) / (n() + l))

   #Estimate genres effect (b_g)
  b_g <- train_data %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_i, by = "movieId") %>%
    mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
    left_join(b_t, by = "day") %>%
    mutate(b_t = replace_na(b_t, 0)) %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - b_t - mu) / (n() + l),
              se = sd(rating) / sqrt(n())) %>%
    filter(se <= s_e)

  # Making predictions on the test set
  predicted_ratings <- test_data %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
    left_join(b_t, by = "day") %>%
    mutate(b_t = replace_na(b_t, 0)) %>%
    left_join(b_g, by = "genres") %>%
    mutate(b_g = replace_na(b_g, 0)) %>%
    mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
    pull(pred)

  # Calculate RMSE and store in the results dataframe
  rmse <- RMSE(predicted_ratings, test_data$rating)
  regularization_results_df <- rbind(regularization_results_df,
                                     data.frame(Lambda = l, RMSE = rmse))
}


# Plot Lambda values vs RMSE
lamda_plot <- ggplot(regularization_results_df, aes(Lambda, RMSE)) +
  geom_point() +
  ggtitle("Lambda Plot")

print(lamda_plot)

})
```



Store lamda and update model

```
suppressWarnings({

model_5_rmse <- min(regularization_results_df$RMSE)

#store optimal lambda
min_lambda <- 5

rmse_results_df <- bind_rows(rmse_results_df,
                          data_frame(Model="Regularised Movie + User + Time + Genre Effects",
                                     RMSE = model_5_rmse ))
print.data.frame(rmse_results_df)

})
```

```
##                                            Model      RMSE
## 1                                      Mean Only 1.0519097
## 2                                   Movie Effect 0.9413492
## 3                            Movie + User Effect 0.8669997
## 4                     Movie + User + Time Effect 0.8665766
## 5                   Movie + User + Genres Effect 0.8669257
## 6 Regularised Movie + User + Time + Genre Effects 0.8628622
```

Regularized movie + user + time + genre effects model generates lowest rmse yet.

Re-estiamte effects with optimal lambda that we previously established

```r
suppressMessages({

# Re-estimate effects with optimal lambda
mu <- mean(train_data$rating)

# Estimate movie effect (b_i)
b_i <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + min_lambda))

# Estimate user effect (b_u)
b_u <- train_data %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu) / (n() + min_lambda))

# Estimate time effect (b_t)
b_t <- train_data %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_i, by = "movieId") %>%
  mutate(time = as_datetime(timestamp)) %>%
  group_by(day = floor_date(time, "day")) %>%
  summarize(b_t = sum(rating - b_i - b_u - mu) / (n() + min_lambda))

# Estimate genres effect (b_g)
b_g <- train_data %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_i, by = "movieId") %>%
  mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
  left_join(b_t, by = "day") %>%
  mutate(b_t = replace_na(b_t, 0)) %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - b_t - mu) / (n() + min_lambda),
            se = sd(rating) / sqrt(n())) %>%
  filter(se <= s_e)

# Predict ratings on the test set using the optimal lambda
predicted_ratings <- test_data %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
  left_join(b_t, by = "day") %>%
  mutate(b_t = replace_na(b_t, 0)) %>%
  left_join(b_g, by = "genres") %>%
  mutate(b_g = replace_na(b_g, 0)) %>%
  mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
  pull(pred)

# Calculate RMSE on the test set
test_rmse <- RMSE(predicted_ratings, test_data$rating)

# Print the RMSE on the test set
print(paste("Test RMSE with optimal lambda:", test_rmse))
})
```

```
## [1] "Test RMSE with optimal lambda: 0.862862239648625"
```

Our rmse further improves with inclusion of the optimal lamda

---

We may now want to use model and tuning parameters to make final predictions on the validation data

```
suppressMessages({

mu <- mean(edx$rating)

# Estimate movie effect (b_i) on the entire dataset
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mean(edx$rating)) / (n() + min_lambda))

# Estimate user effect (b_u) on the entire dataset
b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - first(b_i) - mean(edx$rating)) / (n() + min_lambda))

# Estimate time effect (b_t) on the entire dataset
b_t <- edx %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_i, by = "movieId") %>%
  mutate(time = as_datetime(timestamp)) %>%
  group_by(day = floor_date(time, "day")) %>%
  summarize(b_t = sum(rating - first(b_i) - first(b_u) - mean(edx$rating)) / (n() + min_lambda))

# Estimate genres effect (b_g) on the entire dataset
b_g <- edx %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_i, by = "movieId") %>%
  mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
  left_join(b_t, by = "day") %>%
  mutate(b_t = replace_na(first(b_t), 0)) %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - first(b_i) - first(b_u) - b_t - mean(edx$rating)) / (n() + min_lambda),
            se = sd(rating) / sqrt(n())) %>%
  filter(se <= s_e)


})
```

Make predictions on validation set

```
suppressMessages({



validation_predictions <- final_holdout_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(day = floor_date(as_datetime(timestamp), "day")) %>%
  left_join(b_t, by = "day") %>%
  mutate(b_t = replace_na(first(b_t), 0)) %>%
  left_join(b_g, by = "genres") %>%
  mutate(b_g = replace_na(first(b_g), 0)) %>%
  mutate(pred = mean(final_holdout_test$rating) + first(b_i) + first(b_u)
         + first(b_t) + first(b_g)) %>%
  pull(pred)

# Evaluate the predictions on the validation set
validation_rmse <- RMSE(validation_predictions, final_holdout_test$rating)

# Print the RMSE on the validation set
print(paste("Validation RMSE:", validation_rmse))

})
```

```
## [1] "Validation RMSE: 1.13656473400883"
```

Our final rmse value when making predictions on the validation set is too high. This may suggest overfitting, meaning our models perform well on the training data, but do not perform well on the validation data. Therefore, we may consider other approaches.

Another approach that we learned about is Matrix Factorization

---

Matrix Factorization

```
suppressWarnings({

#install and load the recosystem and float packages

# Install and load the recosystem package
if (!requireNamespace("recosystem", quietly = TRUE)) {
  install.packages("recosystem")
}
library(recosystem)

# Install and load the float package
if (!requireNamespace("float", quietly = TRUE)) {
  install.packages("float", repos = "https://cloud.r-project.org/")
}
library(float)

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

train_reco <- with(train_data, data_memory(user_index = userId, item_index = movieId, rating = rating))
#This line creates a training dataset train_reco in a format suitable for the recommender system.

#It extracts the user IDs, movie IDs, and ratings from the train_data dataset and organizes them into a

test_reco <- with(test_data, data_memory(user_index = userId, item_index = movieId, rating = rating))
```

```
#: Similar to the previous step, this line creates a test dataset test_reco in the #appropriate format 

r <- Reco()
#This line initializes a recommender system object r.

para_reco <- r$tune(train_reco, opts = list(dim = c(20, 30),
                                            costp_l2 = c(0.01, 0.1),
                                            costq_l2 = c(0.01, 0.1),
                                            lrate = c(0.01, 0.1),
                                            nthread = 4,
                                            niter = 10))

#This line tunes the parameters of the recommender system using the training data #train_reco. It searc


r$train(train_reco, opts = c(para_reco$min, nthread = 4, niter = 30))
#After tuning, this line trains the recommender system using the optimal parameters #identified in the 

results_reco <- r$predict(test_reco, out_memory())
# This line uses the trained recommender system to predict ratings for the test dataset test_reco and s

################
factorization_rmse <- RMSE(results_reco, test_data$rating)
rmse_results_df <- bind_rows(rmse_results_df,
                             data.frame(Model = "Matrix factorization",
                                        RMSE = factorization_rmse))
print.data.frame(rmse_results_df)

###############validation

# Convert final_holdout_test to reco format
final_holdout_reco <- with(final_holdout_test, data_memory(user_index = userId, item_index = movieId, r

# Predict ratings on the final_holdout_test set
validation_results_reco <- r$predict(final_holdout_reco, out_memory())

# Calculate RMSE on the final_holdout_test set
validation_factorization_rmse <- RMSE(validation_results_reco, final_holdout_test$rating)

# Add results to the dataframe
rmse_results_df <- bind_rows(rmse_results_df,
                             data.frame(Model = "Matrix factorization on validation set",
                                        RMSE = validation_factorization_rmse))

# Print the updated results
print.data.frame(rmse_results_df)

})
```

```
## iter      tr_rmse          obj
##    0       1.0861   3.3942e+06
##    1       0.8881   2.4445e+06
##    2       0.8659   2.3056e+06
##    3       0.8398   2.1766e+06
##    4       0.8200   2.0835e+06
##    5       0.8028   2.0106e+06
##    6       0.7879   1.9518e+06
##    7       0.7742   1.9013e+06
##    8       0.7614   1.8586e+06
##    9       0.7495   1.8201e+06
##   10       0.7384   1.7866e+06
##   11       0.7283   1.7578e+06
##   12       0.7190   1.7292e+06
##   13       0.7108   1.7062e+06
##   14       0.7032   1.6847e+06
##   15       0.6965   1.6657e+06
##   16       0.6904   1.6498e+06
##   17       0.6847   1.6337e+06
##   18       0.6798   1.6211e+06
##   19       0.6751   1.6084e+06
##   20       0.6709   1.5969e+06
##   21       0.6671   1.5872e+06
##   22       0.6634   1.5767e+06
##   23       0.6601   1.5690e+06
##   24       0.6572   1.5613e+06
##   25       0.6542   1.5535e+06
##   26       0.6516   1.5470e+06
##   27       0.6491   1.5407e+06
##   28       0.6468   1.5344e+06
##   29       0.6446   1.5292e+06
##                                          Model      RMSE
## 1                                    Mean Only 1.0519097
## 2                                Movie Effect 0.9413492
## 3                         Movie + User Effect 0.8669997
## 4                  Movie + User + Time Effect 0.8665766
## 5                Movie + User + Genres Effect 0.8669257
## 6 Regularised Movie + User + Time + Genre Effects 0.8628622
## 7                        Matrix factorization 0.8208978
##                                          Model      RMSE
## 1                                    Mean Only 1.0519097
## 2                                Movie Effect 0.9413492
## 3                         Movie + User Effect 0.8669997
## 4                  Movie + User + Time Effect 0.8665766
## 5                Movie + User + Genres Effect 0.8669257
## 6 Regularised Movie + User + Time + Genre Effects 0.8628622
## 7                        Matrix factorization 0.8208978
## 8          Matrix factorization on validation set 0.8224060
```

Table that stores final results

```
suppressMessages({

library(gt)
gt(rmse_results_df)

})
```

Model	RMSE
Mean Only	1.0519097
Movie Effect	0.9413492
Movie + User Effect	0.8669997
Movie + User + Time Effect	0.8665766
Movie + User + Genres Effect	0.8669257
Regularised Movie + User + Time + Genre Effects	0.8628622
Matrix factorization	0.8208978
Matrix factorization on validation set	0.8224060

Results

Using matrix factorization, while taking much longer, allowed us to achieve a rmse value that is below the desired threshold of 0.86490. The RMSE of 0.82 suggests that, on average, the difference between the actual ratings given by users and the ratings predicted by the recommendation system is approximately 0.82 units.

Conclusion / Future Directions

In summary, this project documented different approaches toward building a machine learning-based movie recommendation system. Ultimately, using matrix factorization led to the lowest rmse value, suggesting the best model fit. In future analyses we may consider further approaches such as principal component analyses or singular value decomposition. However, these procedures will likely take much longer to run and may therefore be challenging to use. Further, we may also consider additional fine-tuning of hyperparameters. For instance, we could reconsider our regularization parameters and experiment with different values such as lasso and ridge regularization to control complexity and prevent overfitting. Additionally we could also consider other models splits, or cross-validation strategies such as k-fold cross-validation or leave-one-out cross-validation. Through systematically exploring and tuning hyperparameters we may further optimize our rmse value and improve model generalization.