Movie Lens Project

Roxie Trachtenberg

2023-12-06

Introduction

According to the Edx project description, the `movielens` data is pulled from the `dslabs` R package and contains a subset of a much larger dataset with millions of movie ratings. This version of the dataset contains 10 million observations with the following variables:

- **userId** is a discrete variable that contains a unique identifier for each user, or movie rater. Note that there can be more than one of the same userId in this dataset if they rate more than one movie or if they rate the same movie more than once at a different time.
- **movieId** is a also a discrete variable that contains a unique identifier for each movie. Note that there can be more than one of the same movieId in this dataset, representing multiple ratings for the same movie from different users at different times.
- **rating** is a discrete variable indicating the number of "stars" awarded to the movie by a specific user at a certain time. Note that there are 11 distinct values that can be awarded, from 0 - 5 and incremented by 0.5.
- **timestamp** is a time and date identifier of when the movie rating was submitted. This could be converted to a more readable date format using a separate R package.
- **title** contains a character string of the movie title and year in a readable format.
- **genres** contains a character string of the main genre(s) of the particular movie in question.

The goal of this project is to build upon concepts and code we have learned throughout the course, and specifically in Course 8, to predict movie ratings from the other features in the dataset. In order to achieve the lowest root mean-squared error (RMSE) possible, multiple techniques, including running a random forest model and utilizing regularization, were performed.

Methods and Analysis

Data Loading and Cleaning

Note that no data cleaning was completed beyond what was provided as part of the project introductory information and all code below has been provided by project staff. This is a reiteration for clarity and continuity.

First, all libraries necessary to complete our analyses were loaded:

```
# Load libraries
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)
library(stringr)
library(readr)
library(dplyr)
library(dslabs)
library(data.table)
library(ranger)
library(splitstackshape)
library(randomForest)

options(timeout = 120)
```

Then, the data was downloaded. merged, and class/column names slightly adjusted to be more comprehensive and readable:

```r
# Load data set from grouplens website
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")

# transform the column classes
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# add movie data and assign clear column names
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")

# transform column classes
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# join movies and ratings data frames to generate the movielens dataset
movielens <- left_join(ratings, movies, by = "movieId")
```

Finally, the final_holdout_test dataset (10% of the movielens dataset) was generated in order to provide a final test of our best model RMSE at the end and the remaining edx set was joined back together:

```r
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## 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)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in 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 test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

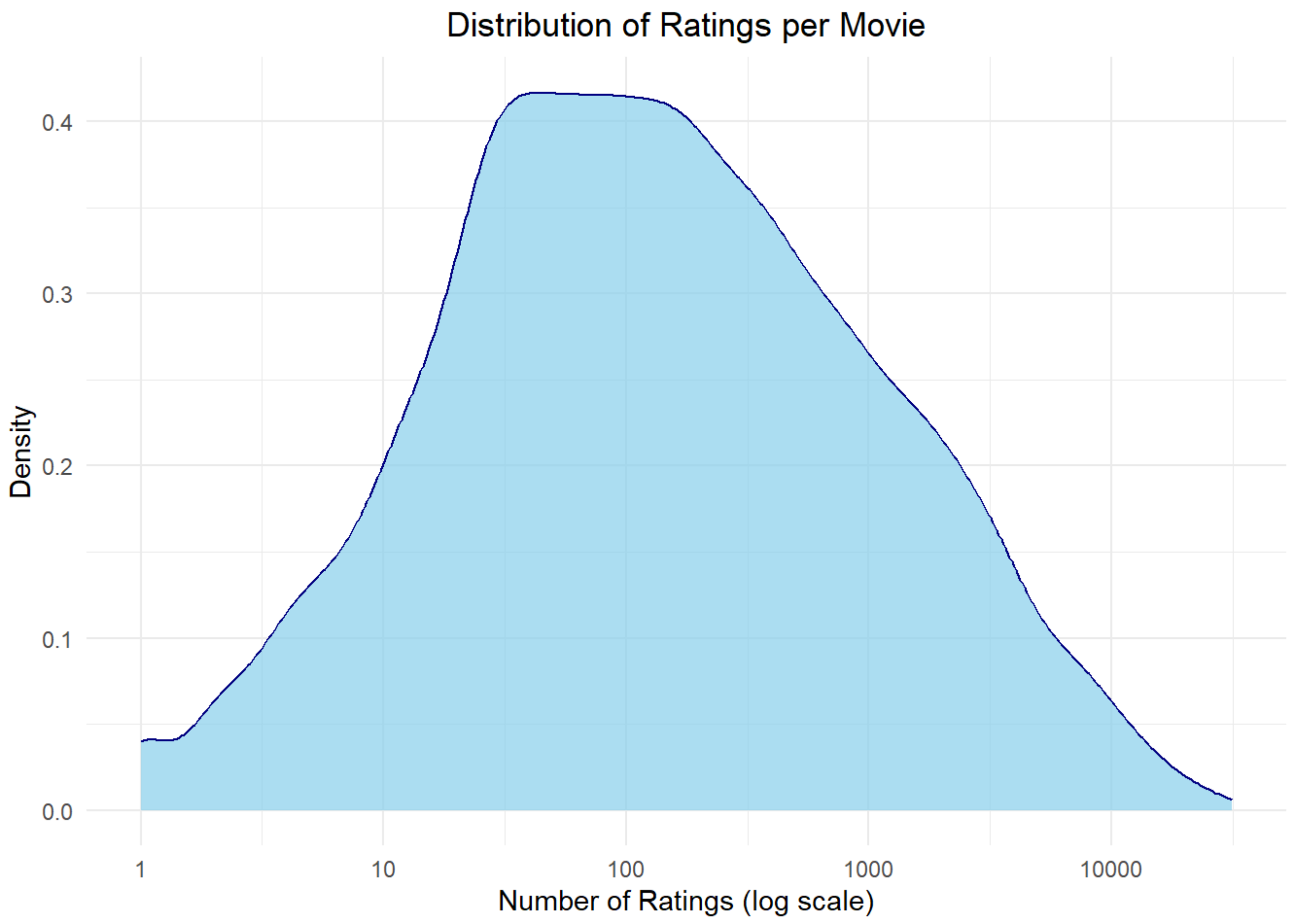
```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```
edx <- rbind(edx, removed)

# remove variables from memory no longer needed
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Exploratory Data Analysis

First, we should explore the distribution of the number of ratings by movieId:

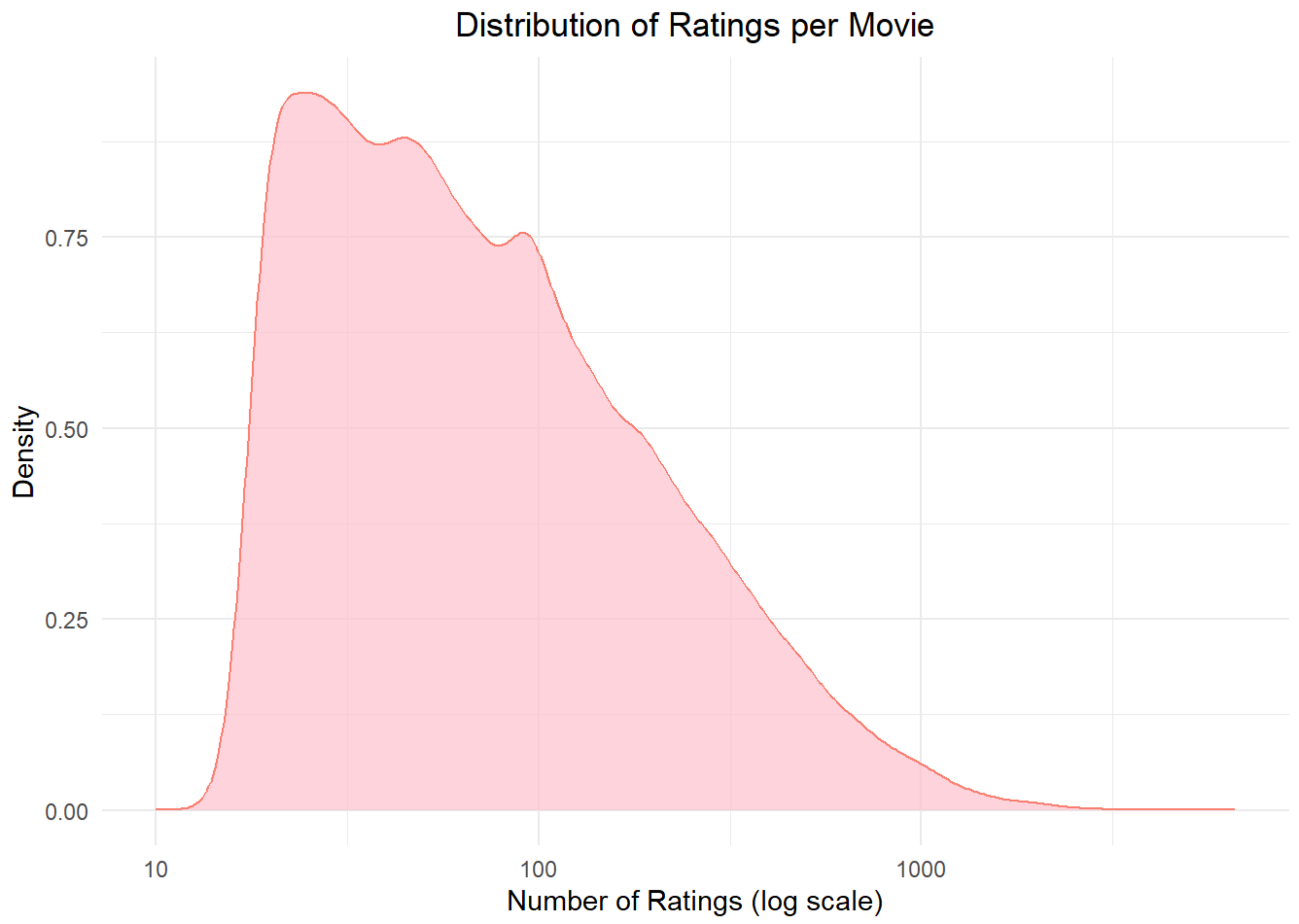
```
ggplot(edx %>% count(movieId), aes(x = n)) +
  geom_density(fill = "skyblue", color = "navy", alpha = 0.7) +
  scale_x_log10() +
  labs(title = "Distribution of Ratings per Movie",
       x = "Number of Ratings (log scale)",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



From the plot, it looks like ratings are relatively log-normally distributed across all movies. This makes sense since this is a large dataset with millions of observations, and the Central Limit Theorem tells us that the sampling distribution of the mean will always be normally distributed, as long as the sample size is large enough.

Next, let's look at distribution of number of ratings per userId:

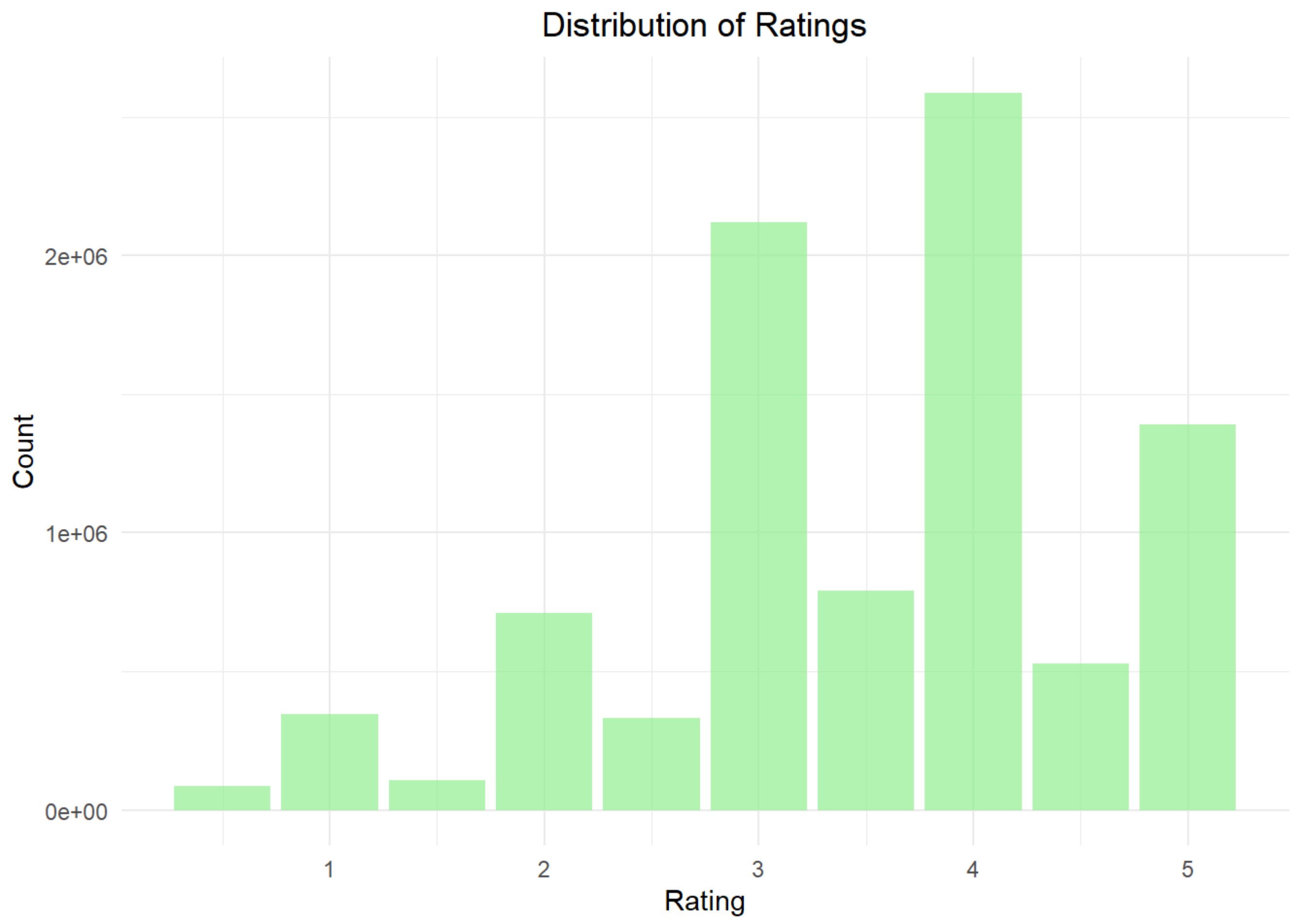
```
ggplot(edx %>% count(userId), aes(x = n)) +
  geom_density(fill = "pink", color = "salmon", alpha = 0.7) +
  scale_x_log10() +
  labs(title = "Distribution of Ratings per Movie",
       x = "Number of Ratings (log scale)",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



This density plot is showing a relatively right skewed distribution of ratings per user, showing that a majority of users submitted less or a medium amount of ratings to the data. This makes sense since only a small subset are likely to be movie critics or consistent reviewers.

Now, let’s explore which discrete ratings were most popular among users:

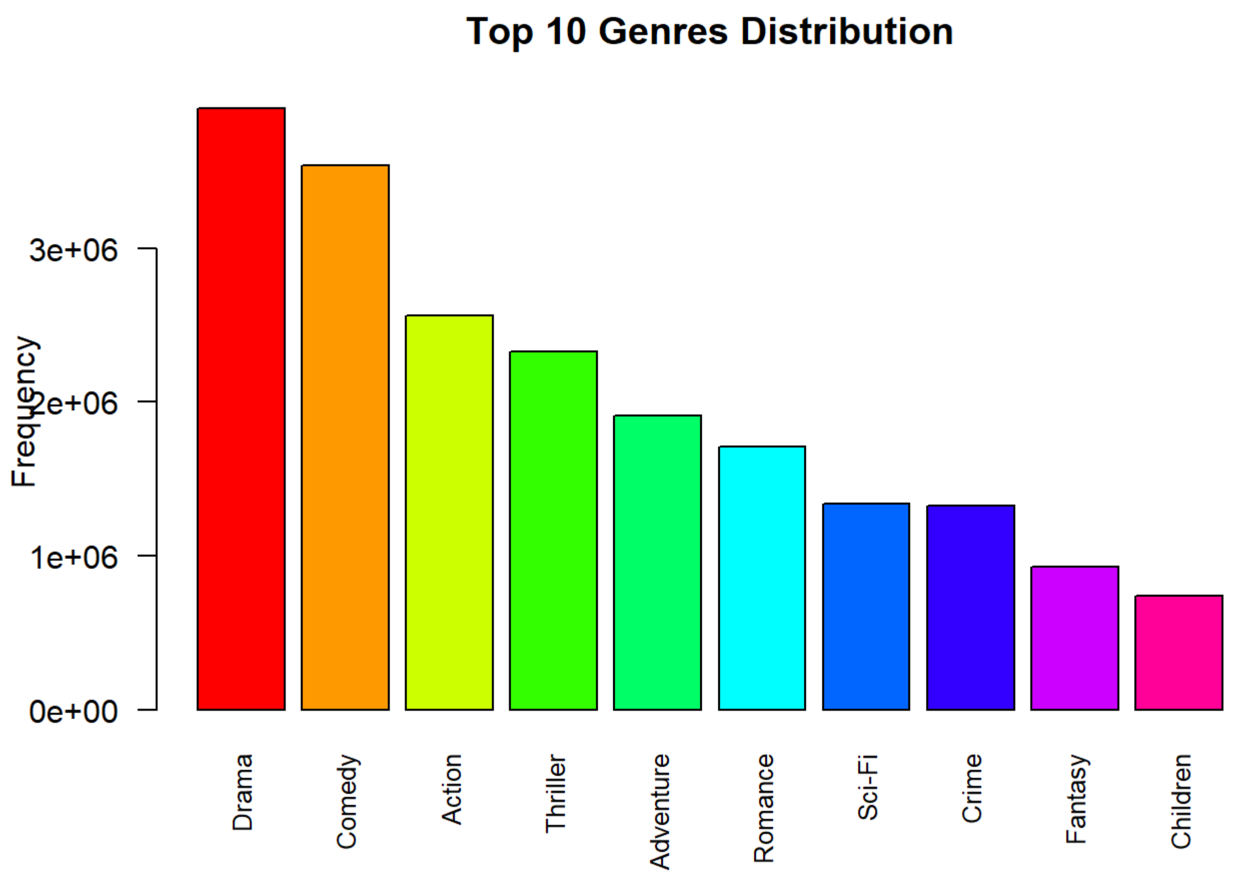
```
ggplot(edx %>% group_by(rating) %>% summarize(count = n()), aes(x = rating, y = count)) +
  geom_bar(stat = "identity", fill = "lightgreen", alpha = 0.7) +
  labs(title = "Distribution of Ratings",
       x = "Rating",
       y = "Count") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



Seems like a rating of “4” is most popular and whole number ratings tend to be more popular than half ratings.

Now, since there are so many distinct genre groups in the dataset (due to the possibility of individual plus combinations of genres) let’s see if we can split the data and visualize the top 10 genres:

```
# One-hot encode the data and split the genre into one per row
# (adding more rows as needed to accommodate the data) for ease of understanding the graph
genres <- cSplit(edx, "genres", sep = "|", direction = "long")

# Exclude "(no genres listed)" category
genres <- filter(genres, genres != "(no genres listed)")

# Get the top 10 genres
top_genres <- names(sort(table(genres$genres), decreasing = TRUE)[1:10])

# Filter the dataset to include only the top 10 genres
genres_top10 <- filter(genres, genres %in% top_genres)

# Order the levels of the factor based on frequency
genres_top10$genres <- factor(genres_top10$genres, levels = names(sort(table(genres_top10$genre
s), decreasing = TRUE)))

# Create a bar plot
barplot(table(genres_top10$genres), col = rainbow(10),
        main = "Top 10 Genres Distribution", ylab = "Frequency",
        las = 2, cex.names = 0.8)  # las = 2 for vertical labels, adjust cex.names for label siz
e
```



It looks like Drama is our top genre, with Comedy close behind in terms of number of ratings. However, note that one movie may occupy two to several genres.

Modeling

Data Preparation

To start, I split the edx dataset into training and test sets and defined the RMSE function that we will use to assess our models going forward:

```
# First, split edx dataset into test and train sets

test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

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

Basic Movie Effects Model

Let’s start by running code provided by instructors to get an idea of what a simple model would yield if we just considered movie effect in addition to overall mean rating:

```
# start with the mean of all ratings
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu_hat)
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
mu <- mean(train_set$rating)

# generate the movie effect model
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results # show table
```

```
## # A tibble: 2 × 2
##   method              RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Movie Effect Model 0.944
```

Movie + User Effects Model

Now, as evidenced previously from course code, we can add in the user effects model:

```
# movie and user effect model
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# make predictions using new model and print RMSE to table
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
rmse_results # show updated table
```

```
## # A tibble: 3 × 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Just the average           1.06
## 2 Movie Effect Model         0.944
## 3 Movie + User Effects Model 0.866
```

Movie + User + Genre Effects Model

The addition of user effect yields a slightly better RMSE, but let’s see if we can do even better by building on course code and adding in the genre-specific effect:

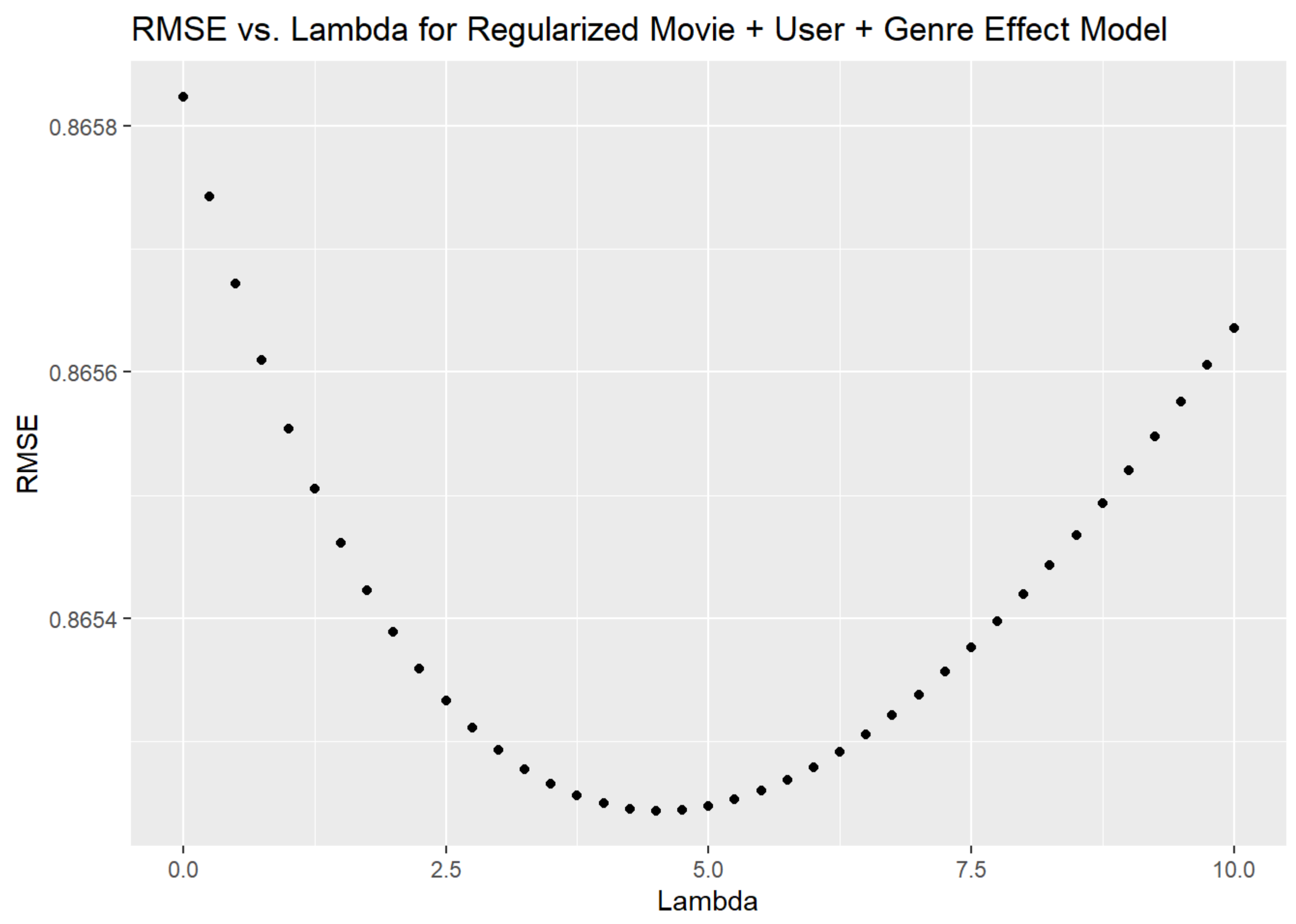
```r
# genre effect model
genre_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

# make predictions using new model and print RMSE to table
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Genre Effects Model",
                                     RMSE = model_3_rmse ))
rmse_results # show updated table
```

```
## # A tibble: 4 × 2
##   method                              RMSE
##   <chr>                              <dbl>
## 1 Just the average                   1.06
## 2 Movie Effect Model                 0.944
## 3 Movie + User Effects Model         0.866
## 4 Movie + User + Genre Effects Model 0.866
```

Regularized Movie + User + Genre Effects Model

Now let’s see if we can improve our model by incorporating regularization:

```
# Apply regularization to the movie + user + genre model
# WARNING: This code will take at least 5 mins to run depending on machine capabilities

# Regularization parameter search for Movie + User + Genre Effect Model
lambdas <- seq(0, 10, 0.25)

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

for (lambda in lambdas) {
  # Movie effect model with regularization
  movie_avgs_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

  # User effect model with regularization
  user_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))

  # Genre effect model with regularization
  genre_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    left_join(user_avgs_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  # Predict on the test set
  predicted_ratings <- test_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    left_join(user_avgs_reg, by = "userId") %>%
    left_join(genre_avgs_reg, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  # Calculate RMSE
  model_rmse <- RMSE(predicted_ratings, test_set$rating)

  # Store the results
  rmse_results_lambda <- bind_rows(rmse_results_lambda,
                                   data.frame(Lambda = lambda, RMSE = model_rmse))
}

# Plot RMSE vs. lambda
qplot(Lambda, RMSE, data = rmse_results_lambda) + geom_point() +
  labs(title = "RMSE vs. Lambda for Regularized Movie + User + Genre Effect Model")
```



```
# Find the lambda with the minimum RMSE
lambda_min <- rmse_results_lambda$Lambda[which.min(rmse_results_lambda$RMSE)]

# Display the optimal lambda
cat("Optimal Lambda:", lambda_min, "\n")
```

```
## Optimal Lambda: 4.5
```

```
# Update rmse_results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Regularized Movie + User + Genre Effect Model",
                                     RMSE = min(rmse_results_lambda$RMSE)))
rmse_results # show updated table
```

```
## # A tibble: 5 × 2
##   method                                         RMSE
##   <chr>                                         <dbl>
## 1 Just the average                              1.06
## 2 Movie Effect Model                            0.944
## 3 Movie + User Effects Model                    0.866
## 4 Movie + User + Genre Effects Model            0.866
## 5 Regularized Movie + User + Genre Effect Model 0.865
```

This seems to give a similar result to the previous model, indicating that regularization may have helped slightly, but overall, not significantly.

Random Forest using Subsetted Data

Let’s try an ensemble machining learning method, using random forest and a subset of the training data (due to data processing time and machine memory and processing capabilities):

```
# random forest model
# Define the Random Forest model and use a subset of the data for training due to runtime issues
subset_train_set <- as.data.table(train_set)[1:10000, ]

rf_model <- randomForest(rating ~ userId + movieId + genres, data = subset_train_set, ntree = 5)

# Print the summary of the Random Forest model
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = rating ~ userId + movieId + genres, data = subset_train_set,      ntr
ee = 5)
##                Type of random forest: regression
##                      Number of trees: 5
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 1.177786
##                     % Var explained: -9.04
```

```
# Make predictions on the test set
predicted_ratings_rf <- predict(rf_model, newdata = test_set)

# Calculate RMSE
rf_model_rmse <- RMSE(predicted_ratings_rf, test_set$rating)
rf_model_rmse
```

```
## [1] 1.153695
```

```
# Update rmse_results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Random Forest on Subsetted Data",
                                     RMSE = rf_model_rmse))
rmse_results
```

```
## # A tibble: 6 × 2
##   method                                        RMSE
##   <chr>                                        <dbl>
## 1 Just the average                             1.06 
## 2 Movie Effect Model                           0.944
## 3 Movie + User Effects Model                   0.866
## 4 Movie + User + Genre Effects Model           0.866
## 5 Regularized Movie + User + Genre Effect Model 0.865
## 6 Random Forest on Subsetted Data              1.15 
```

This is an interesting exercise in that it gives an RMSE worse than that of our model showing just the rating average. Instead of running a more comprehensive random forest model (that would likely take multiple hours to run) with a larger portion or the full dataset, we can try to use techniques such as cross-validation to run and evaluate our model.

Cross-Validation

Finally, let's try cross-validation. This will help us split the data into more manageable chunks to run the model on.

```
# WARNING: This code could take up to ~30 minutes to run, pending your machine capabilities. I h
ave set the "eval = " argument in this code chunk of the rmd file to "FALSE" so that it will not
run if the reviewer does not wish to run it in the rmd file itself.

# Set the number of folds for cross-validation
num_folds <- 5  # You can adjust the number of folds as needed

# Create an empty vector to store cross-validated predictions
cv_predictions <- numeric(length = nrow(train_set))
cv_rmse <- 0  # Initialize RMSE

# Perform cross-validation
for (fold in 1:num_folds) {
  # Create training and validation sets for the current fold
  set.seed(fold)  # Ensure reproducibility across folds
  fold_indices <- createDataPartition(y = train_set$rating, p = 0.8, list = FALSE)
  train_fold <- train_set[fold_indices, ]
  val_fold <- train_set[-fold_indices, ]

  # Train the random forest model using ranger
  rf_model <- ranger(rating ~ userId + movieId, data = train_fold, num.trees = 50)

  # Make predictions on the validation set
  fold_predictions <- predict(rf_model, data = val_fold)$predictions

  # Store the predictions in the cv_predictions vector
  cv_predictions[-fold_indices] <- fold_predictions

  # Update RMSE
  cv_rmse <- cv_rmse + RMSE(fold_predictions, val_fold$rating)
}

# Calculate average RMSE across folds
cv_rmse <- cv_rmse / num_folds

cat("Cross-validated RMSE:", cv_rmse, "\n")

rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Cross Validated Model",
                                     RMSE = cv_rmse))
rmse_results
```

When this code ran previously, I got an RMSE value of 0.953, so in case you didn’t want to take 20-30 minutes to run the code above, I have added this rmse value to the table for simplicity and ease of evaluation. Set the below code chunk to eval = FALSE in the rmd file if you would like to run the code above instead.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Cross Validated Model",
                                     RMSE = 0.953))
rmse_results
```

```
## # A tibble: 7 × 2
##   method                                         RMSE
##   <chr>                                         <dbl>
## 1 Just the average                              1.06 
## 2 Movie Effect Model                            0.944
## 3 Movie + User Effects Model                    0.866
## 4 Movie + User + Genre Effects Model            0.866
## 5 Regularized Movie + User + Genre Effect Model 0.865
## 6 Random Forest on Subsetted Data               1.15 
## 7 Cross Validated Model                         0.953
```

This was a good experiment and way to see if we could fit smaller portions of the data and aggregate results (which we clearly can!), but the RMSE is less performant with this cross-validation method compared to the other RMSEs achieved with previous models, so we will likely run our final_holdout_test on a model that achieved a better RMSE with the test_set.

Results

First, let’s inspect a sorted version of our RMSE summary table:

```
rmse_results_sorted <- rmse_results %>% arrange(RMSE)
rmse_results_sorted
```

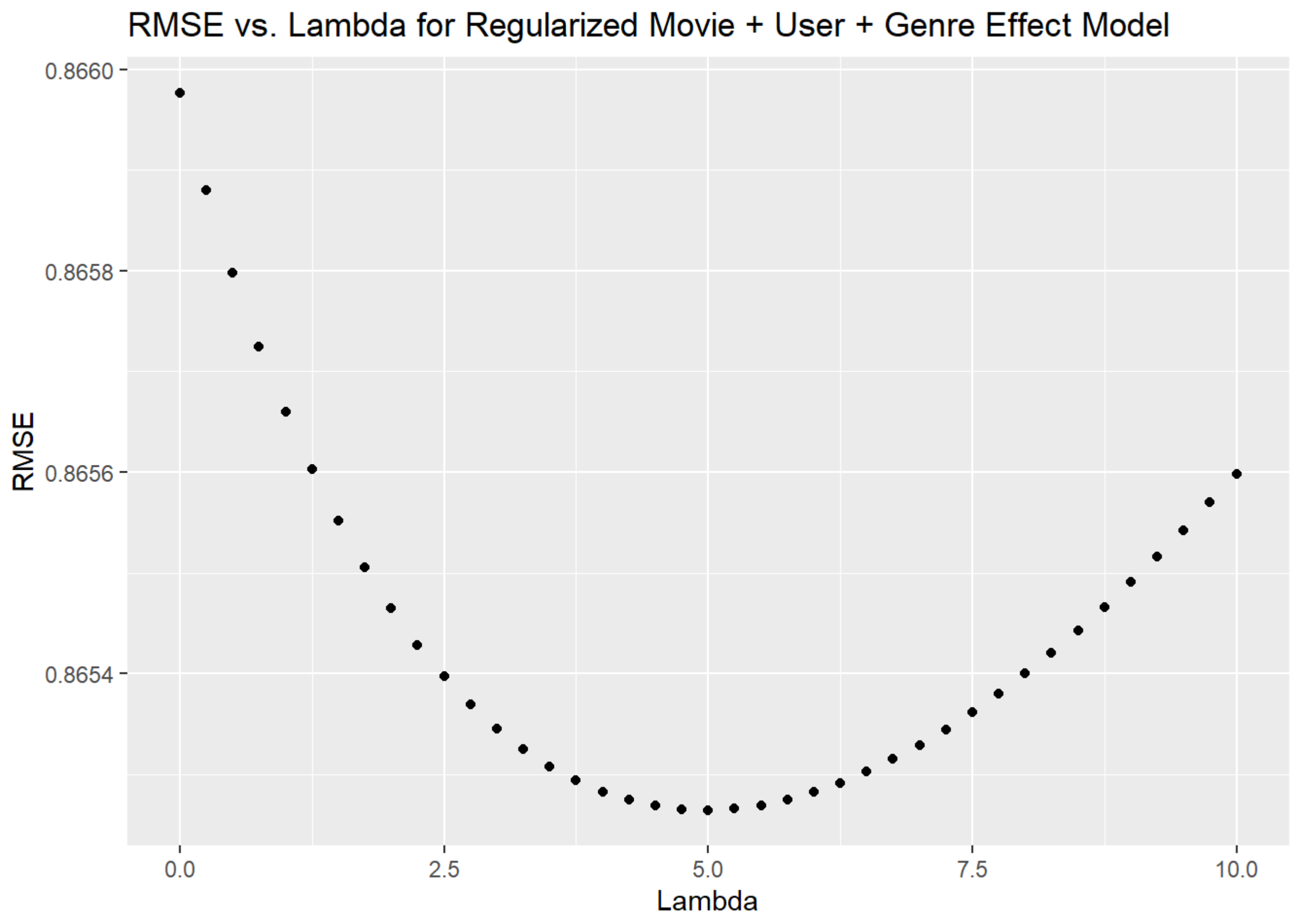
```
## # A tibble: 7 × 2
##   method                                         RMSE
##   <chr>                                         <dbl>
## 1 Regularized Movie + User + Genre Effect Model 0.865
## 2 Movie + User + Genre Effects Model            0.866
## 3 Movie + User Effects Model                    0.866
## 4 Movie Effect Model                            0.944
## 5 Cross Validated Model                         0.953
## 6 Just the average                              1.06 
## 7 Random Forest on Subsetted Data               1.15 
```

Based on the rmse table, let’s run our final_holdout_test on the best-performing model, the regularized movie + user + genre effect model:

```r
# Let's test our best model against the holdout_test_set to achieve our final RMSE:

# Regularization parameter search for Movie + User + Genre Effect Model
lambdas <- seq(0, 10, 0.25)

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

for (lambda in lambdas) {
  # Movie effect model with regularization
  movie_avgs_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

  # User effect model with regularization
  user_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))

  # Genre effect model with regularization
  genre_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    left_join(user_avgs_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  # Predict on the final holdout test
  predicted_ratings <- final_holdout_test %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    left_join(user_avgs_reg, by = "userId") %>%
    left_join(genre_avgs_reg, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  # Filter out rows with missing values
  valid_rows <- !is.na(predicted_ratings)
  predicted_ratings <- predicted_ratings[valid_rows]
  final_holdout_ratings <- final_holdout_test$rating[valid_rows]

  # Calculate RMSE
  model_rmse <- RMSE(predicted_ratings, final_holdout_ratings)

  # Print RMSE value
  cat("Lambda:", lambda, "RMSE:", model_rmse, "\n")

  # Store the results
  rmse_results_lambda <- bind_rows(rmse_results_lambda,
                                   data.frame(Lambda = lambda, RMSE = model_rmse))
}
```

```
## Lambda: 0 RMSE: 0.8659768
## Lambda: 0.25 RMSE: 0.8658802
## Lambda: 0.5 RMSE: 0.8657975
## Lambda: 0.75 RMSE: 0.8657248
## Lambda: 1 RMSE: 0.8656602
## Lambda: 1.25 RMSE: 0.8656027
## Lambda: 1.5 RMSE: 0.8655513
## Lambda: 1.75 RMSE: 0.8655054
## Lambda: 2 RMSE: 0.8654647
## Lambda: 2.25 RMSE: 0.8654286
## Lambda: 2.5 RMSE: 0.8653969
## Lambda: 2.75 RMSE: 0.8653692
## Lambda: 3 RMSE: 0.8653452
## Lambda: 3.25 RMSE: 0.8653248
## Lambda: 3.5 RMSE: 0.8653077
## Lambda: 3.75 RMSE: 0.8652937
## Lambda: 4 RMSE: 0.8652826
## Lambda: 4.25 RMSE: 0.8652743
## Lambda: 4.5 RMSE: 0.8652686
## Lambda: 4.75 RMSE: 0.8652654
## Lambda: 5 RMSE: 0.8652645
## Lambda: 5.25 RMSE: 0.8652658
## Lambda: 5.5 RMSE: 0.8652693
## Lambda: 5.75 RMSE: 0.8652747
## Lambda: 6 RMSE: 0.8652821
## Lambda: 6.25 RMSE: 0.8652913
## Lambda: 6.5 RMSE: 0.8653022
## Lambda: 6.75 RMSE: 0.8653147
## Lambda: 7 RMSE: 0.8653289
## Lambda: 7.25 RMSE: 0.8653445
## Lambda: 7.5 RMSE: 0.8653615
## Lambda: 7.75 RMSE: 0.86538
## Lambda: 8 RMSE: 0.8653997
## Lambda: 8.25 RMSE: 0.8654206
## Lambda: 8.5 RMSE: 0.8654428
## Lambda: 8.75 RMSE: 0.8654661
## Lambda: 9 RMSE: 0.8654905
## Lambda: 9.25 RMSE: 0.8655159
## Lambda: 9.5 RMSE: 0.8655424
## Lambda: 9.75 RMSE: 0.8655698
## Lambda: 10 RMSE: 0.8655981
```

```
# Plot RMSE vs. lambda
qplot(x = Lambda, y = RMSE, data = rmse_results_lambda) + geom_point() +
  labs(title = "RMSE vs. Lambda for Regularized Movie + User + Genre Effect Model")
```



```
# Find the lambda with the minimum RMSE
lambda_min <- rmse_results_lambda$Lambda[which.min(rmse_results_lambda$RMSE)]

# Display the optimal lambda
cat("Optimal Lambda:", lambda_min, "\n")
```

```
## Optimal Lambda: 5
```

```
# Display RMSE
final_rmse = min(rmse_results_lambda$RMSE)
round(final_rmse, 5)
```

```
## [1] 0.86526
```

We have achieved a final RMSE of 0.86526.

Conclusion

Overall, we achieved a decent RMSE using the regularized movie + user + genre effect model, but I think there were many opportunities here for improvement in our data pre-processing and modeling approach.

First and foremost, I think this dataset required much more cleaning and refining than was completed in my project. Although NAs were addressed in most of the models run, because there are so many observations, the data could have benefitted from some strategic subsetting, such as removing rows where no genre was listed, analyzing the timestamps and removing any rows showing years prior to a reasonable date range for the data, and removing observations showing major outliers in any of the variables. Also, I did not explore effects of the “timestamp” variable on rating, which could have provided valuable information and improved RMSE in certain models. This would have been relatively straightforward to implement and transform this variable to a more usable and readable format.

In addition, using sparse matrices and one-hot encoding of genre variables to run in random forest models could have provided a more robust model for us to use. It was a challenge to run efficient models without overwhelming my machine and taking hours of runtime to test models, so further exploration of how to strategically subset and run this random forest model in a more efficient way could have been beneficial. In addition, tuning hyperparameters of the random forest model could have had the potential to significantly improve those results. This is something I plan to explore and implement in a different context as part of the choose-your-own capstone project process.

Finally, just like experimenting with the hyperparameters in random forest would have likely helped the results, experimenting with different fold numbers and parameters in the cross-validated model could have also likely improved results. In addition, using bootstrapping methods could have been a good strategy to explore as a modeling technique.