MovieLens Project

Martin J Page

07/06/2020

Executive Summary

In this project, we create a movie recommendation system that predicts the ratings of movies based on the user, movie, genre, and release year variables. The data are from the MovieLens dataset of 10,000,054 ratings applied to 10,681 movies by 71,567 users of the online movie recommender service MovieLens. Users were selected at random for inclusion. All users had rated at least 20 movies.

We train a machine learning algorithm using the inputs in the train set (`edx`) to predict movie ratings in the validation set (`validation`). We evaluate the movie rating predictions of the model compared to the true ratings using RMSE. The model is build step-by-step, with each iteration accounting for more potential effects (i.e. user effects, movie effects, genre effects, release year effects, and regularisation). The train set (`edx`) is divided into a training set (`training`) and a testing set (`testing`). The `training` set is used to build the model and the `testing` set is used to perform intermediate evaluations of the model. The `validation` set is not used to train the algorithm (intermediate evaluation with the `validation` set would effectively train the model on this data!). The `validation` set is used only for the final RMSE evaluation.

Based on the predicted movie rating that a user would give a specific movie, the system can recommend movies that it predicts the user would rate highly (i.e. enjoy watching).

Methods

See the R script file `MovieLens_Script.R` for the detailed code used in the methods.

Setting up the Data

Loading the Data The data were downloaded from the weblink http://files.grouplens.org/datasets/movielens/ml-10m.zip and stored in a temporary ZIP file. The ZIP file contains two datasets, one for the ratings and one for the movies. These were loaded separately and joined to make one data frame.

```
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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
```

```
movielens <- left_join(ratings, movies, by = "movieId")
save(movielens, file = "movielens.RData")
```

Formatting the Data The `genres` variable was formatted as a factor with 797 levels. These levels are unique combinations of 20 primary genres. It was decided not to split the genre combinations (e.g. using `dplyr::separate_rows(movielens, genres, sep = "\\|")`) as the specific genre combinations might have a unique effect on the ratings that would be lost if they were separated into the constituent genres. A column of the release `year` was added by extracting the year indicated at the end of each movie `title`. A column `date` was added, which converts the timestamp into the date a rating was posted.

```
movielens$genres <- factor(movielens$genres)
movielens$year <- as.numeric(str_sub(movielens$title, start = -5, end = -2))
movielens$date <- as_datetime(movielens$timestamp)
```

Data Partitions

Partitioning the Train Data and the Validation Data

The full `movielens` data set was partitioned into train set (`edx`) at 90% and a validation set (`validation`) at 10%. Only `edx` is used to train the models, whereas `validation` is used to evaluate only the final model. The seed is set to ensure reproducibility. Because not every user rated every movie, the `validation` set is checked to ensure that it contains users and movies that also appear in `edx` (we can only evaluate predictions if we known the true rating). Any non-matching observations are put back into `edx` to train the models.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%  semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")
removed <- anti_join(temp, validation); edx <- rbind(edx, removed)
```

In order to perform intermediate evaluations of the model, we also partition the train set (`edx`) further into a `training` and `testing` set. The `training` set is used to systematically build a more complex model that considers more variables. After each step of model building, we use the `testing` set to evaluate the intermediate model. Importantly, any intermediate testing that is performed on the `validation` set could lead to overfitting. Model building decisions are made based on the RMSE evaluatuions, and using the `validation` set to inform these decision is effectively using the `validation` data to train the model. As previously, we set the seed and ensure that the observations in the `testing` set are represented in the `training` set.

```
set.seed(2, sample.kind = "Rounding")
inTest <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
training <- edx[-inTest,]
temp_test <- edx[inTest,]

testing <- temp_test %>% semi_join(training, by = "movieId") %>%
  semi_join(training, by = "userId")
removed_test <- anti_join(temp_test, testing); training <- rbind(training, removed_test)
```

RMSE Function

We define the RMSE calculation used to evaluate the models. This is the square root of the mean of the squared differences between the true and predicted ratings. The value represents the typical error we make when predicting a movie rating. An error of 1 in this context means an uncertainty equal to a one star rating.

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

Building Models

Model 1: The Average of all Movies

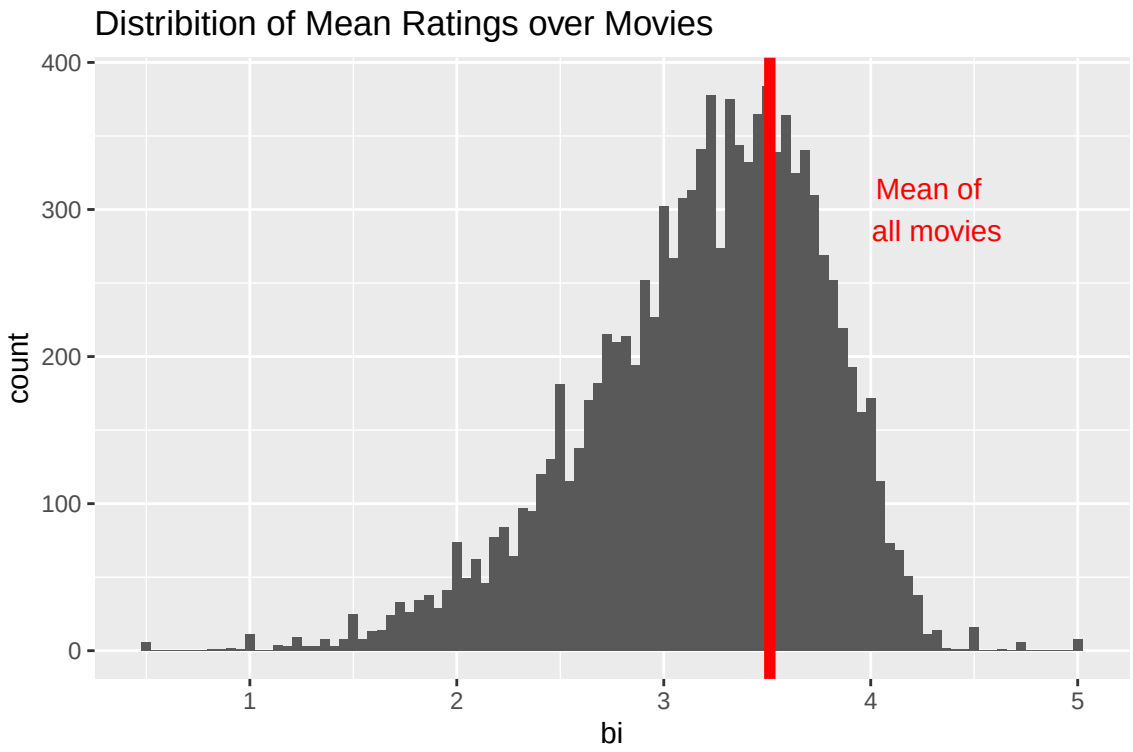
As the most simplistic model, we predict the rating of a movie to be just the average rating of all the movies (general rating).

```
mu_hat <- mean(training$rating)
model_1_rmse <- RMSE(testing$rating, mu_hat)
rmse_results <- tibble(method = "Average Rating (Naive)", RMSE = model_1_rmse)
```

Model 2: Model Movie Effects

To improve on our prediction based on the general (naive) rating, we consider that each movie will have its own average rating.

```
training %>% group_by(movieId) %>% summarise(bi = mean(rating)) %>% ggplot(aes(bi)) +
  geom_histogram(bins = 100) + geom_vline(xintercept = mu_hat, lwd = 2, colour = "red") +
  annotate("text", label = "Mean of \n all movies", x = 4.3, y = 300, colour = "red") +
  ggtitle("Distribition of Mean Ratings over Movies")
```



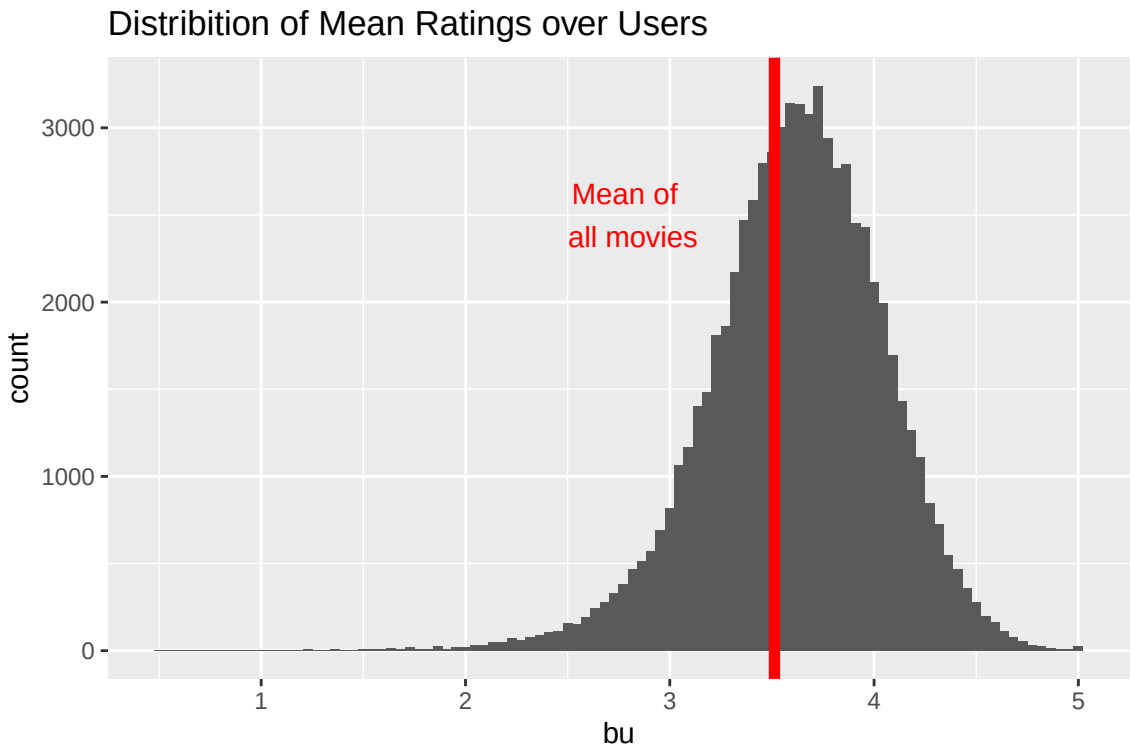
We therefore find the average rating for each movie, adding a so-called 'movie effect' into our model as a term that adjusts the general rating for each specific movie.

```
movie_avg <- training %>% group_by(movieId) %>% summarise(b_i = mean(rating - mu_hat))
pred_bi <- testing %>% left_join(movie_avg, by = "movieId") %>%
    mutate(pred = mu_hat + b_i) %>%.$pred
model_2_rmse <- RMSE(pred_bi, testing$rating)
rmse_results <- bind_rows(rmse_results,
            tibble(method = "Movie Effect Model", RMSE = model_2_rmse))
```

Model 3: Model User Effects

Each user also has their own average rating behaviour.

```
training %>% group_by(userId) %>% summarise(bu = mean(rating)) %>% ggplot(aes(bu)) +
  geom_histogram(bins = 100) + geom_vline(xintercept = mu_hat, lwd = 2, colour = "red") +
  annotate("text", label = "Mean of \n all movies", x = 2.8, y = 2500, colour = "red") +
  ggtitle("Distribition of Mean Ratings over Users")
```



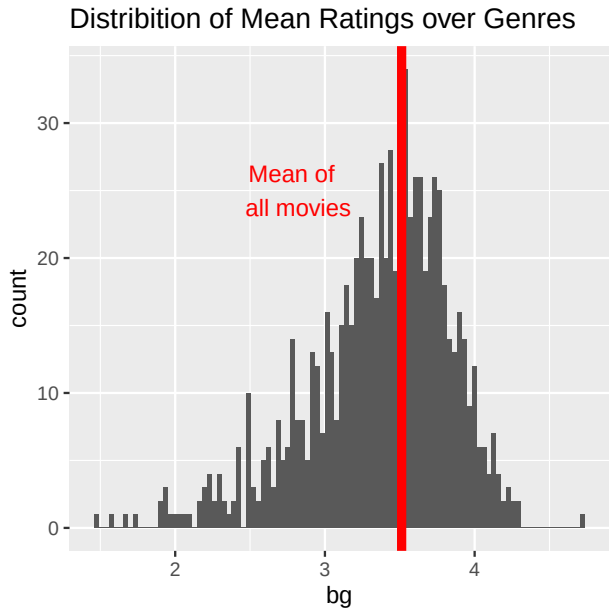
To improve on our model that considers that each movie will have a unique average rating, we include a term that accounts for a similar phenomenon with users: each user with have a unique average rating that they give to movies.

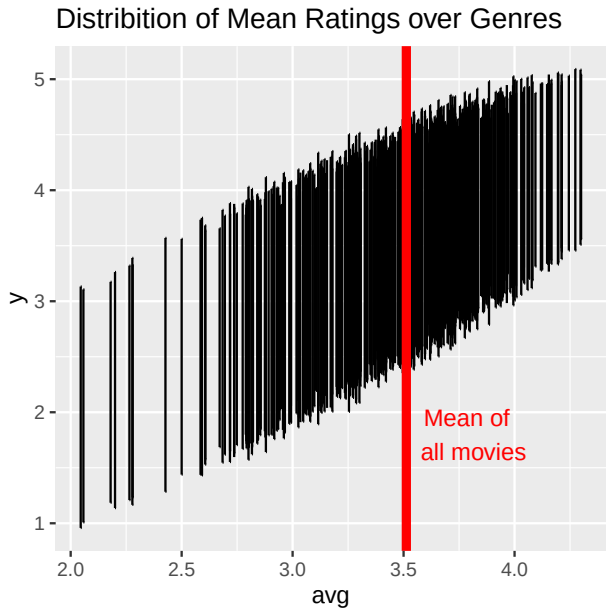
```
user_avg <- training %>% left_join(movie_avg, by = "movieId") %>%
    group_by(userId) %>% summarise(b_u = mean(rating - mu_hat - b_i))
pred_bi_bu <- testing %>% left_join(movie_avg, by = "movieId") %>%
    left_join(user_avg, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>% .$pred
model_3_rmse <- RMSE(pred_bi_bu, testing$rating)
rmse_results <- bind_rows(rmse_results,
            tibble(method = "Movie + User Effects Model", RMSE = model_3_rmse))
```

Model 4: Genre Effects

Genres are another factor that exibit different average ratings for each group.

```
p1 <- training %>% group_by(genres) %>% summarise(bg = mean(rating)) %>% ggplot(aes(bg)) +
  geom_histogram(bins = 100) + geom_vline(xintercept = mu_hat, lwd = 2, colour = "red") +
  annotate("text", label = "Mean of \n all movies", x = 2.8, y = 25, colour = "red") +
  ggtitle("Distribition of Mean Ratings over Genres")
p2 <- training %>% group_by(genres) %>% filter(n() > 1000) %>%
  summarise(avg = mean(rating), sd = sd(rating)) %>%
  arrange(avg) %>% ggplot() + geom_errorbar(aes(x = avg, ymin = avg - sd, ymax = avg + sd)) +
  geom_vline(xintercept = mu_hat, lwd = 2, colour = "red") +
  annotate("text", label = "Mean of \n all movies", x = 3.8, y = 1.8, colour = "red") +
  ggtitle("Distribition of Mean Ratings over Genres")
grid.arrange(p1,p2, ncol = 2)
```





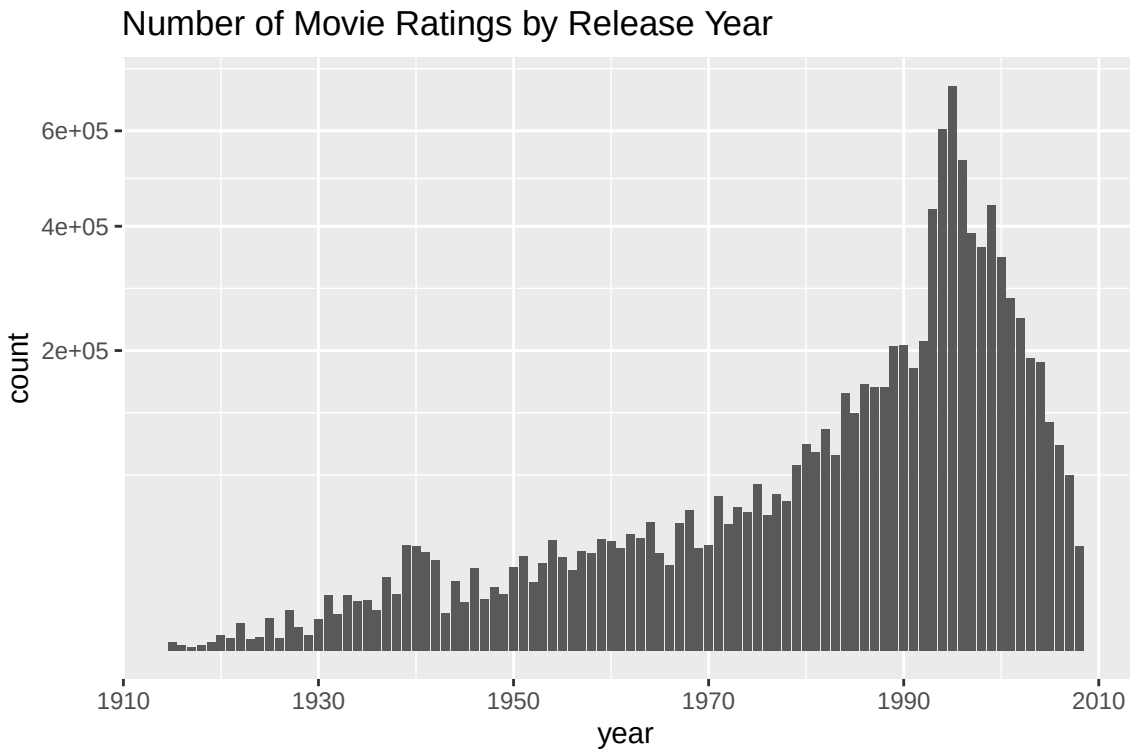
As a next step, we model the relationship that the genre has on ratings by including a term in the model that adjusts the predicted rating based on the average rating for each genre combination.

```
genre_avg <- training %>% left_join(movie_avg, by = "movieId") %>%
    left_join(user_avg, by = "userId") %>%
    group_by(genres) %>% summarise(b_g = mean(rating - mu_hat - b_i - b_u))
pred_bi_bu_bg <- testing %>% left_join(movie_avg, by = "movieId") %>%
    left_join(user_avg, by = "userId") %>% left_join(genre_avg, by = "genres") %>%
    mutate(pred = mu_hat + b_i + b_u + b_g) %>% .$pred
model_4_rmse <- RMSE(pred_bi_bu_bg, testing$rating)
rmse_results <- bind_rows(rmse_results,
            tibble(method = "Movie + User + Genres Effects Model", RMSE = model_4_rmse))
```

Model 5: Release Year Effects

A final variable we shall consider is the release year. Older movies will have had a longer time to accummulate ratings but their might also be a possible difference in ratings between newer, modern and much older movies.

```
p3 <- training %>%  group_by(movieId) %>%  ggplot(aes(x = year)) + geom_bar() +
  scale_y_continuous(trans = "sqrt") + ggtitle("Number of Movie Ratings by Release Year")
p3
```



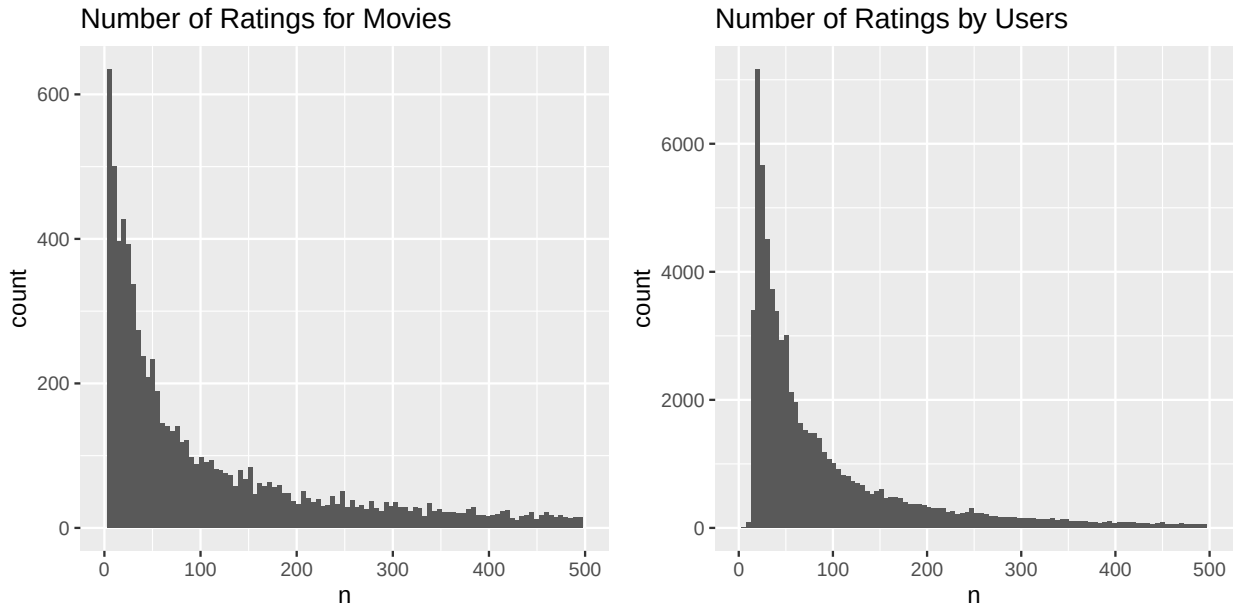
As a final layer of the model, we add a term that considers the average rating for each year of release.

```
year_avg <- training %>% left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  left_join(genre_avg, by = "genres") %>%
    group_by(year) %>% summarise(b_r = mean(rating - mu_hat - b_i - b_u - b_g))
pred_bi_bu_bg_br <- testing %>% left_join(movie_avg, by = "movieId") %>%
    left_join(user_avg, by = "userId") %>% left_join(genre_avg, by = "genres") %>%
    left_join(year_avg, by = "year") %>%
    mutate(pred = mu_hat + b_i + b_u + b_g + b_r) %>% .$pred
model_5_rmse <- RMSE(pred_bi_bu_bg_br, testing$rating)
rmse_results <- bind_rows(rmse_results,
 tibble(method = "Movie + User + Genres + Release Year Effects Model", RMSE = model_5_rmse))
```

Model 6: Regularisaion

Different movies will have a different number of ratings. Similarly, diffrent users rate more actively than others. Movies with ratings from very few users will have more uncertainty. This logic applies to all the variables in our model.

```
p4 <- training %>% count(movieId) %>% ggplot(aes(n)) + geom_histogram(bins = 100) +
    scale_x_continuous(limits = c(0,500)) + ggtitle("Number of Ratings for Movies")
p5 <- training %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(bins = 100) +
   scale_x_continuous(limits = c(0,500)) + ggtitle("Number of Ratings by Users")
grid.arrange(p4,p5, ncol = 2)
```



Large errors will increase the RMSE. We therefore apply regularisation to each variable in the model (movie, user, genre and release year effects) to penalise (shrink) large estimates that are computed using small sample sizes. The regularistion approach is coded as a function `rmse_reg_fun` so that it can be reused. We test a few lambda values (the penalty) on the sample size (n) to establish the one that minimises the RMSE, using `sapply()`.

```
rmse_reg_fun <- function(training, testing, l){
    mu <- mean(training$rating)
    b_i <- training %>% group_by(movieId) %>%
        summarise(b_i = sum(rating - mu)/(n()+l))
    b_u <- training %>% left_join(b_i, by = "movieId") %>%
        group_by(userId) %>% summarise(b_u = sum(rating - b_i - mu)/(n()+l))
    b_g <- training %>% left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        group_by(genres) %>% summarise(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
    b_r <- training %>% left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>% left_join(b_g, by ="genres") %>%
        group_by(year) %>% summarise(b_r = sum(rating - b_i - b_u - b_g - mu)/(n()+l))

    predicted_ratings <- testing %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_g, by ="genres") %>%
        left_join(b_r, by = "year") %>%
        mutate(pred = mu + b_i + b_u + b_g + b_r) %>%
        .$pred

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

lambdas <- seq(0, 10, 0.25)
rmses_reg <- sapply(lambdas, rmse_reg_fun, training = training, testing = testing)
model_6_rmse <- min(rmses_reg)
rmse_results <- bind_rows(rmse_results,
            tibble(method = "Regularised Model (Final Model)", RMSE = model_6_rmse))
```

Validation Evaluation

We evalulate the final model that adjusts the general average movie rating by 4 adjustment averages (i.e. movie, user, genre and release year effects) and uses regularisation to penalise (shrink) estimates computed on small sample sizes with the optimal tuning parameter, lambda, from the `training` set.

```
lambda <- lambdas[which.min(rmses_reg)]
validation_rmse <- rmse_reg_fun(training, validation, lambda)
rmse_results <- bind_rows(rmse_results,
            tibble(method = "Final Validation", RMSE = validation_rmse))
```

Results

Model 1

The first model we built predicts the same rating for all movies regardless of the user: the general rating ($\hat{\mu}$). This model therefore assumes that all differences are explained by random variation. The model is given by the equation:
$Y_{u,i} = \mu + \epsilon_{u,i}$

```
rmse_results[1,] %>% knitr::kable()
```

method	RMSE
Average Rating (Naive)	1.06031

Model 2

Different movies are rated differently. The second model adds the term b_i to represent the average ranking for movie i. It effectively adjust the general rating toward the mean rating for the specific movie i. The model is given by the equation:
$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$

```
rmse_results[2,] %>% knitr::kable()
```

method	RMSE
Movie Effect Model	0.9436226

Model 3

To explain more of the variability that our model currently puts to random variation, we consider that different users will also give ratings differently. We add the term b_u to the third model to adjust the general rating further by the mean ratings for user u. The model is given by the equation:
$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

```
rmse_results[3,] %>% knitr::kable()
```

method	RMSE
Movie + User Effects Model	0.8663835

Model 4

We refine the model further by adding a term for the genre effect, $g_{u,i}$, the genre for user's u rating of movie i. The model is given by the equation:
$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k + \epsilon_{u,i}$, with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k

```
rmse_results[4,] %>% knitr::kable()
```

method	RMSE
Movie + User + Genres Effects Model	0.8660658

Model 5

The last variable we add to the model is the influence of release year, $r_{u,i}$, of a movie i on a user's u rating. The model is given by the equation:
$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k + r_{u,i} + \epsilon_{u,i}$, with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k
(note: I'm not sure what notation $r_{u,i}$ should take in the formula.)

```
rmse_results[5,] %>% knitr::kable()
```

method	RMSE
Movie + User + Genres + Release Year Effects Model	0.8658876

Model 6

Finally, we penalise estimates in our model that are calculated using small samples sizes. For regularisation, we minimise the following equation:
$\frac{1}{N} \sum_{u,i} (y_{u,i} - \mu - b_i - b_u - b_g - b_r)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2 + \sum_g b_g^2 + \sum_r b_r^2)$

```
rmse_results[6,] %>% knitr::kable()
```

method	RMSE
Regularised Model (Final Model)	0.8653754

Conclusion

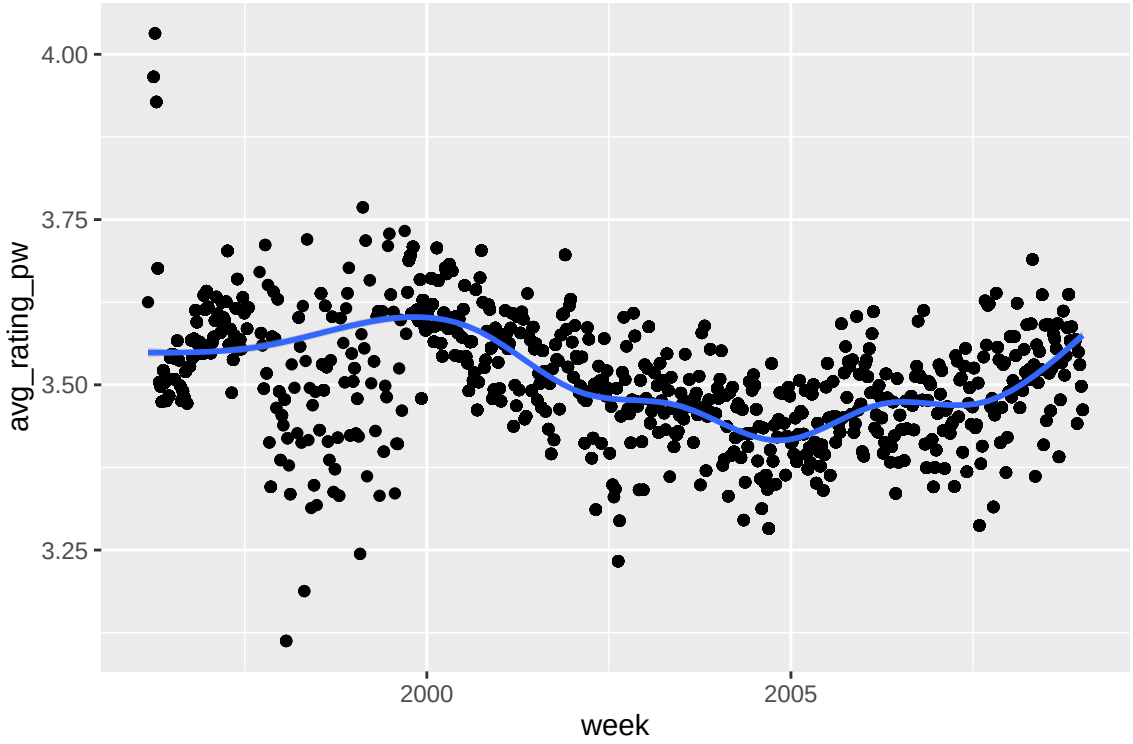
We have a build a model that successfully minimises the RMSE. We did this by systematically adding terms to our prediction algorithm that explained the variability of the data. These terms - namely movie, user, genre, and release year effects - adjust the naive average rating to a prediction rating conditioned on the parameters of each observation. The final check is to evaulate our final model against the `validation` set that is completely untouched until this point. The valdiation score is below the RMSE benchmark value of **0.86490**.. Since the `validation` set was not use to try the model, we are confident that the RMSE is not the result of overtraining.

```
rmse_results %>% knitr::kable() %>% row_spec(row = 7, bold = TRUE)
```

method	RMSE
Average Rating (Naive)	1.0603099
Movie Effect Model	0.9436226
Movie + User Effects Model	0.8663835
Movie + User + Genres Effects Model	0.8660658
Movie + User + Genres + Release Year Effects Model	0.8658876
Regularised Model (Final Model)	0.8653754
Final Validation	**0.8647307**

A further variable we could consider in the model is the `timestamp` of when ratings were submitted by users. Plotting the average rating for each week against the day suggests some evidence of a time effect.

```
p6 <- training %>% mutate(week = round_date(date, unit = "week")) %>% group_by(week) %>%
  mutate(avg_rating_pw = mean(rating)) %>% ungroup() %>% sample_n(size = 10000) %>%
  ggplot(aes(week, avg_rating_pw)) + geom_point() + geom_smooth()
p6
```



We could therefore extend our model further by adding a term $d_{u,i}$ as the day for user's u rating of movie i. The model would be given by the equation:
$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k + r_{u,i} + f(d_{u,i}) + \epsilon_{u,i}$, with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k and f and f a smooth function of $d_{u,i}$

A limitation of the model is that the users are not well described. Users of different ages and genders, or from different regions might exhibit certain patterns when rating movies that could refine our model. In other words, including demographic information about the users could enhance our model. Future work could also attempt more computationally costly models such as k-nearest neighbours, principal component analysis, random forest, bagging, boosting, etc.

Reference:
F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872