MovieLens Project

Yousef Waiel Said

26/12/2023

Contents

1 Overview

This Report is related to the MovieLens Project under HarvardX Data Science Course: PH125.9x. The MovieLens Project in this course requires the creation of a movie recommendation system using the MovieLens dataset. Recommender Systems are systems that seek to predict or filter preferences according to user's choices. The data provided is from the MovieLens 10M set (i.e. 10 million ratings), a much larger version of the data set contained in the dslabs library used during the course. The given data set 'edx' used for training the model has approximately 9 million ratings. The outcome that we need to predict is the rating $Y_{u,i}$ given by the user u for a given movie i. The features that we can use to predict the ratings $Y_{u,i}$ are:

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

The edx dataset will be partitioned into a training and a test set. Different models will be built using the training set. The Movie ratings predictions from these models will be compared to the true ratings in the test subset using RMSE. The model with the lowest RMSE will be chosen to train on the entire edx set and make the final predictions on the validation set. These predictions on the validation set will be compared to the true ratings using Root Mean Squared Error(RMSE).

Least Square Estimates along with Regularisation will be used.

2 Data Cleaning, Analysis and Visualisation

The data given in the edx dataset is already in tidy format (as seen below). Thereby, Data Wrangling of any form is not needed.

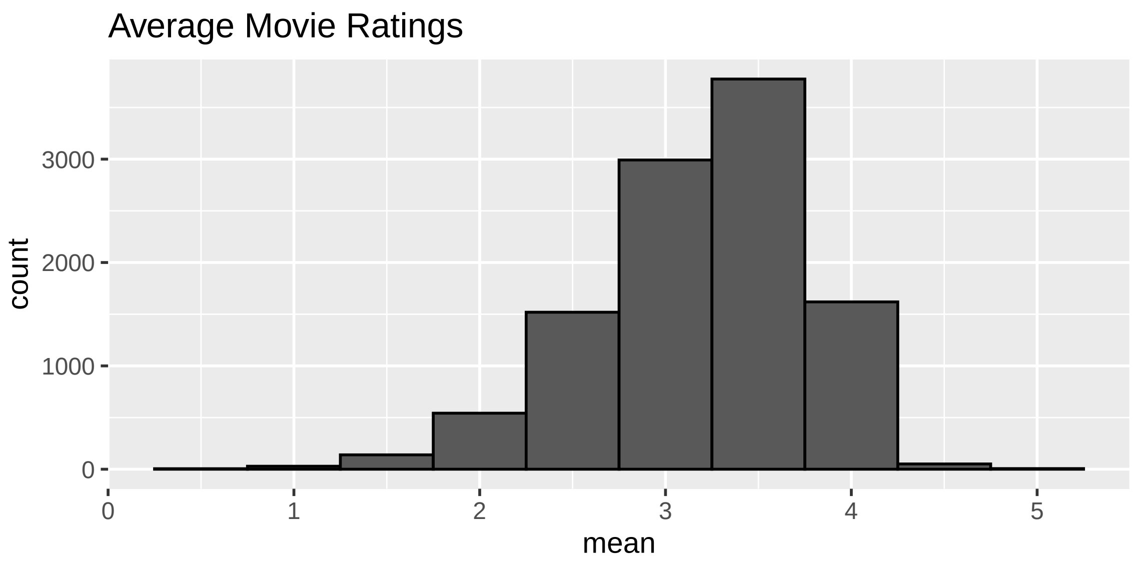
```r
# Displaying the first 6 entries
head(edx) %>%
  knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	NA	NA
1	185	5	838983525	NA	NA
1	292	5	838983421	NA	NA
1	316	5	838983392	NA	NA
1	329	5	838983392	NA	NA
1	355	5	838984474	NA	NA

2.1 Movie Effect

By plotting the average rating of each movie, it can be seen that there is considerable variability among different movies. Some movies are generally rated higher than others, while others are much lower than the average.

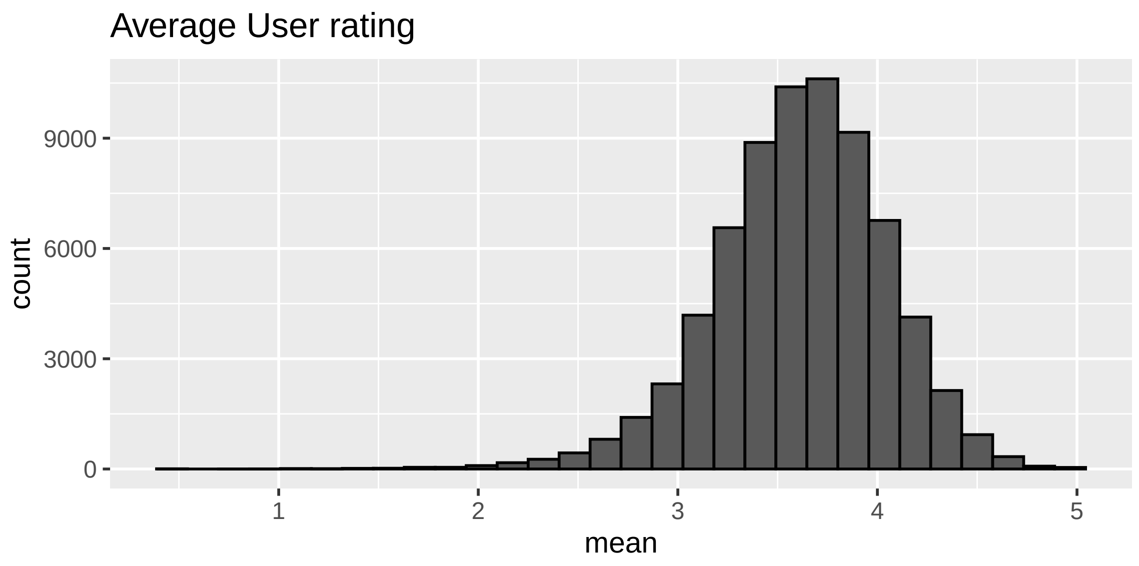
```r
# Group by Movie ID and plot the mean rating for each movie
edx %>% group_by(movieId) %>%
  summarize(mean = mean(rating)) %>%
  ggplot(aes(x=mean)) +
  geom_histogram(bins=10,color=I("black")) +
  ggtitle("Average Movie Ratings")
```



2.2 User Effect

By plotting the average ratings given by each user, it can be seen that there is considerable variability across users as well. Some users give very poor ratings on average, while others give very good reviews.

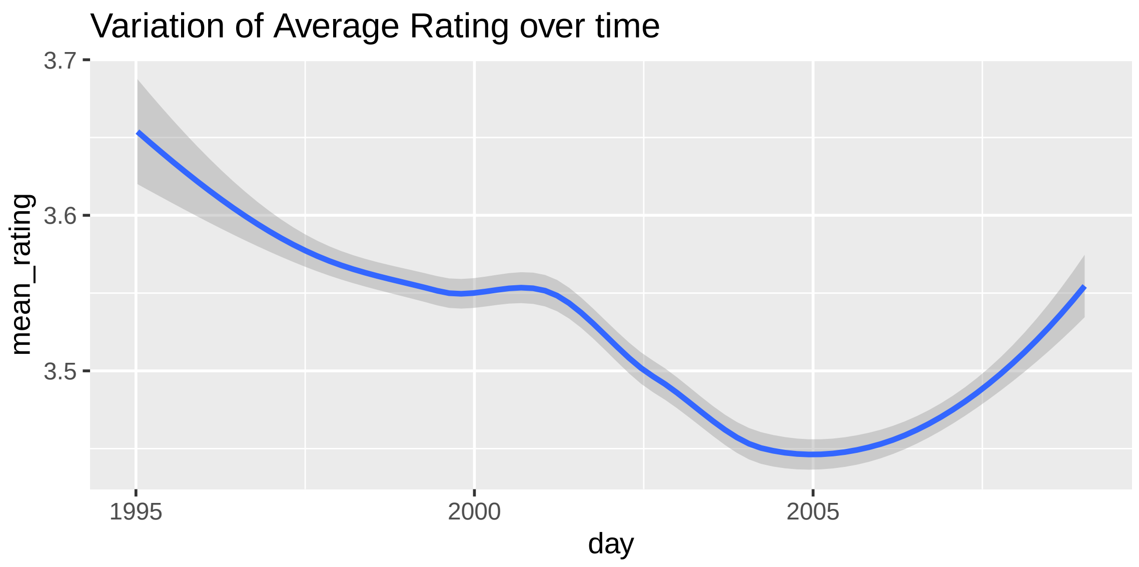
```r
# Group by User ID and plot the mean ratings given by each user
edx %>% group_by(userId) %>%
  summarise(mean=mean(rating)) %>%
  ggplot(aes(mean)) +
  geom_histogram(bins=30,color=I("black")) +
  ggtitle("Average User rating")
```



2.3 Time Effect

On computing and plotting the average rating against each day, it can be seen that time has some effect on the average rating. This could be an indiciation of the interests and habits of society changing over time.

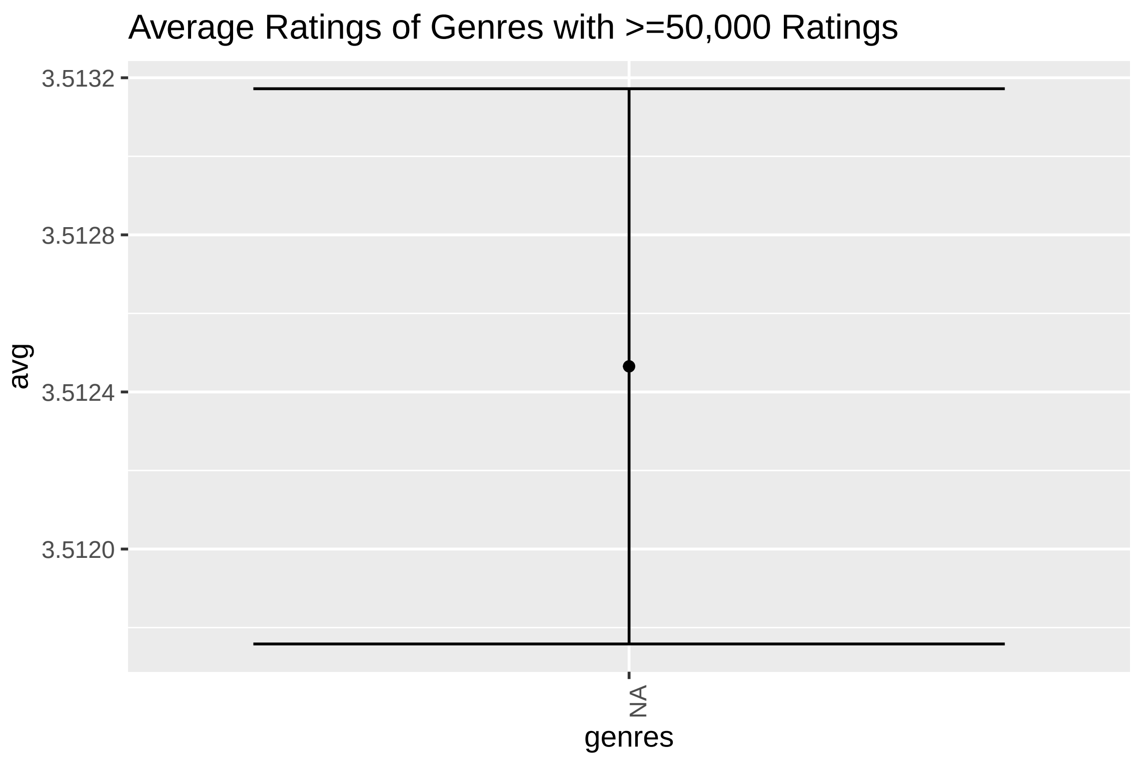
```
# Group ratings by date and plot the smoothed conditional mean over the days
edx %>% mutate(time=as_datetime(timestamp)) %>% # Converting epoch time to human-readable time
  group_by(day=floor_date(time,"day")) %>% # Rounding date-time down to the nearest day
  summarise(mean_rating=mean(rating)) %>%
  ggplot(aes(day,mean_rating)) +
  geom_smooth(method='loess', formula = y~x) +
  ggtitle("Variation of Average Rating over time")
```



2.4 Genre Effect

On plotting error bar plots for average ratings of movies grouped by genres with more than 50,000 ratings, evidence of genre effect is found.

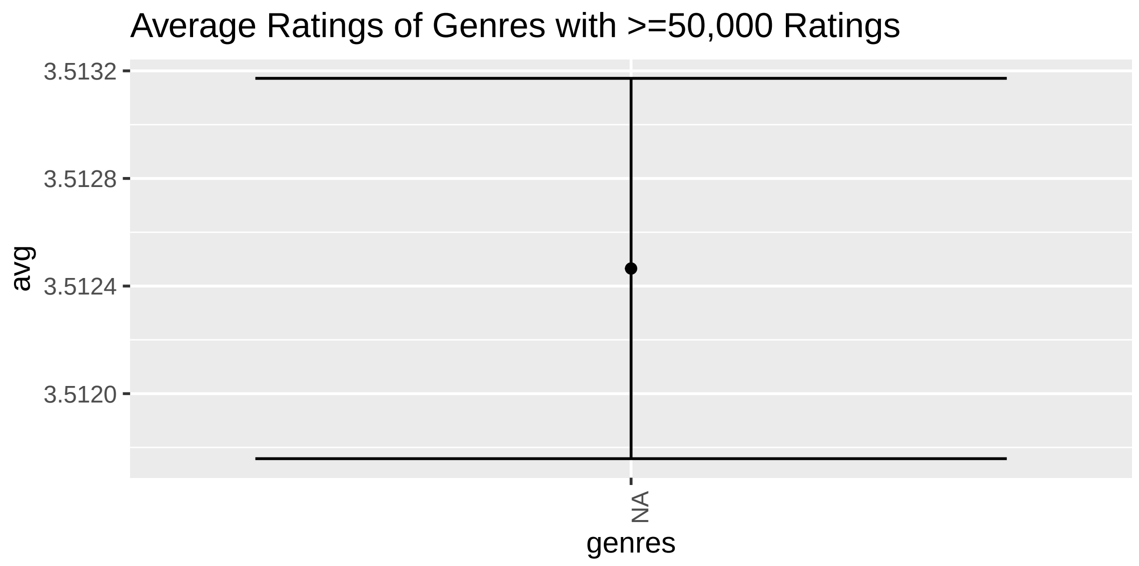
```
# Group ratings by genres and plot the error bar plots for genres with over 50,000 ratings
edx %>% group_by(genres) %>%
  summarize(n=n(),avg=mean(rating),se=sd(rating)/sqrt(n())) %>% # mean and standard errors
  filter(n >= 50000) %>% # Keeping genres with ratings over 50,000
  mutate(genres = reorder(genres, avg)) %>% # order genres by mean ratings
  ggplot(aes(x=genres,y=avg,ymin=avg-2*se,ymax=avg+2*se)) + # lower and upper confidence intervals
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  ggtitle("Average Ratings of Genres with >=50,000 Ratings")
```



Also, on plotting error bar plots for average ratings of movies grouped by genres, regardless of the number of ratings, evidence of genre effect is found.

Some of the genres have large standard errors.

```
# Group ratings by genres and plot the error bar plots for all genres
edx %>% group_by(genres) %>%
  summarize(n=n(),avg=mean(rating),se=sd(rating)/sqrt(n())) %>% # mean and standard errors
  filter(n>1) %>%
  mutate(genres = reorder(genres, avg)) %>% # order genres by mean ratings
  ggplot(aes(x=genres,y=avg,ymin=avg-2*se,ymax=avg+ 2*se)) + # lower and upper confidence intervals
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  ggtitle("Average Ratings of Genres with >=50,000 Ratings")
```



3 Data Modelling

The edx dataset is split into training and test datasets. The train set will be the subset used to train different models. The test set will be the subset used to test and evaluate the trained models.

Once the model has been finalized and the tuning parameters chosen, the entire edx set will be used to train the final model and make predictions on the validation set.

```r
# Create train and test subsets from the edx set
# Test set will be 20% of the edx set
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

3.1 Naive Bayes

The first model assumes the same rating for all movies and all users, with all the differences explained by random variation: If μ represents the true rating for all movies and users and ϵ represents independent errors sampled from the same distribution centered at zero, then:

$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$

In this case, the least squares estimate of μ — the estimate that minimizes the root mean squared error — is the average rating of all movies across all users.

The average rating for the train set is:

```r
# Calculating mean rating
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.512482
```

The residual mean squared error is then computed by comparing the predicted ratings with the true ratings in the test set:

```r
# Calculating RMSE
naive_rmse <- RMSE(test_set$rating, mu)
```

```
naive_rmse
```

```
## [1] 1.059904
```

We see that the baseline RMSE is 1.059904 A table is created to store the RMSE results obtained from different models.

```r
# Creating a data frame to store RMSEs
rmse_results <- data_frame(Method = "Just the average", RMSE = naive_rmse)
print.data.frame(rmse_results)
```

```
##             Method     RMSE
## 1 Just the average 1.059904
```

3.2 Movie Effect

From data exploration and visualisation we had observed that there is evidence of a movie effect. We can improve our model by adding a term, b_i, that represents the average rating for movie i :

$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$

b_i is the average of $Y_{u,i}$ minus the overall mean μ for each movie i.

We can use least squares to estimate the b_i using the lm function. But, becuase there are over ten thousand movies, a b_i needs to be assigned for each movie which would make the lm function very slow here.

In this particular situation, we know that the least square estimate b_i is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i.

So, it is computed this way:

```r
# Computing b_i for each movie as mean of residuals
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

RMSE is then calculated after incorporating the Movie Effect.

```r
# Predicting movie ratings
predicted_ratings <- mu + test_set %>%
  left_join(b_i, by='movieId') %>%
  .$b_i

# Calculating RMSE
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

# Storing the RMSE result in the rmse_results dataframe
rmse_results <- bind_rows(rmse_results,
    data_frame(Method="Movie Effect Model",
    RMSE = model_1_rmse ))

print.data.frame(rmse_results)
```

```
##               Method      RMSE
## 1   Just the average 1.0599043
## 2 Movie Effect Model 0.9437429
```

We see that the RMSE has improved by around 11% to 0.9437429

3.3 User Effect

We also know that there is evidence of a user effect from data exploration and visulatisation done earlier.

We can improve our model by adding another term, b_u, that represents the average rating given by user u:

$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$

where b_u is a user-specific effect

If a user who gives poor ratings on average (negative b_u) rates a great movie (positive b_i), the effects counter each other. We may be able to get close to the correct prediction, thereby reducing the RMSE.

Least squares can again be used to estimate the b_u. But, becuase there are close to seventy thousand users, a b_u needs to be assigned to each user. The lm function would, again, be very slow here.

b_u can instead be estimated as the average of $y_{u,i} - \mu - \hat{b}_i$

```
# Computing b_u for each user as mean of residuals
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/n())

# Predicting movie ratings on test set
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

# Calculating RMSE and Storing the RMSE result in the rmse_results dataframe
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
    data_frame(Method="Movie + User Effects",
    RMSE = model_2_rmse ))
print.data.frame(rmse_results)
```

```
##                 Method      RMSE
## 1     Just the average 1.0599043
## 2   Movie Effect Model 0.9437429
## 3 Movie + User Effects 0.8659320
```

After incorporating the User effects, we see that the RMSE has improved by another 9% to 0.865932

3.4 Time Effect

Through data exploration and visualisation we found that there is some time effect on the movie ratings.

If we define $d_{u,i}$ as the day for user's u rating of movie i, the rating $Y_{u,i}$ can be written as:

$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i}$ with f a smooth function of $d_{u,i}$

The smooth function $f(d_{u,i})$ can be approximated by a least square estimate b_t for each day.

Thus, the rating $Y_{u,i}$ can be re-written as:

$Y_{u,i} = \mu + b_i + b_u + b_t + \varepsilon_{u,i}$

where $t = t_{u,i}$

Again, estimating b_t as the average of $y_{u,i} - \mu - \hat{b}_i - \hat{b}_u$, because of our computational constraints:

```
# Computing b_t for each day as mean of residuals
b_t <- train_set %>%
    left_join(b_u, by="userId") %>%
    left_join(b_i, by = "movieId") %>%
    mutate(time=as_datetime(timestamp)) %>% #Converting epoch time to human readable time
    group_by(day=floor_date(time,"day")) %>% #rounding date-time down to nearest day
    summarise(b_t=sum(rating-b_i-b_u-mu)/n())

# Predicting movie ratings on test set
predicted_ratings <- test_set %>%
```

```r
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(day=floor_date(as_datetime(timestamp),"day")) %>% #rounding date-time down to nearest day
    left_join(b_t, by="day") %>%
    mutate(b_t=replace_na(b_t,0)) %>% #Replacing NA values with 0
    mutate(pred = mu + b_i + b_u + b_t) %>%
   .$pred

# Calculating RMSE and Storing the RMSE result in the rmse_results dataframe
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
    data_frame(Method="Movie + User Effects + Time effect",
    RMSE = model_3_rmse ))
print.data.frame(rmse_results)
```

```
##                              Method      RMSE
## 1                  Just the average 1.0599043
## 2                Movie Effect Model 0.9437429
## 3              Movie + User Effects 0.8659320
## 4 Movie + User Effects + Time effect 0.8654403
```

We can see that including a time effect estimate in the model lowers the RMSE of our predictions to 0.8654403

3.5 Genre Effect

We know from data exploration and visualisation that different genres have different average ratings.

We can improve our model even further by adding adding a least square estimate, b_g, that estimates the effect of each genre:

$Y_{u,i} = \mu + b_i + b_u + b_t + b_g + \varepsilon_{u,i}$

where $t = t_{u,i}$ and $g = g_{u,i}$

Instead of using the lm function, we estimate b_g as the average of residual: $y_{u,i} - \mu - \hat{b}_i - \hat{b}_u - \hat{b}_t$

```r
# Computing b_g for each genre as mean of residuals
b_g <- train_set %>%
    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) %>%
    summarise(b_g=(sum(rating-b_i-b_u-b_t-mu))/(n()))

# Predicting movie ratings on test set
predicted_ratings <- test_set %>%
    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)) %>% #Replacing NA values with 0
    left_join(b_g, by="genres") %>%
    mutate(b_g=replace_na(b_g,0)) %>% #Replacing NA values with 0
    mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
    .$pred

# Calculating RMSE and storing the RMSE result in the rmse_results dataframe
model_4_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
    data_frame(Method="Movie + User + Time + Genre Effects",
    RMSE = model_4_rmse ))
print.data.frame(rmse_results)
```

```
##                                Method      RMSE
## 1                    Just the average 1.0599043
## 2                  Movie Effect Model 0.9437429
## 3                Movie + User Effects 0.8659320
## 4  Movie + User Effects + Time effect 0.8654403
## 5 Movie + User + Time + Genre Effects 0.8654403
```

With a Genre effect included in the model, the RMSE decreases to 0.865104

We had seen from an earlier error bar plot "Average ratings of Different Genres" that some genres have high standard errors. We can filter out genres with high standard errors on their mean ratings to further decrease the RMSE. For genres having high standard errors, it is better to be conservative and not assign a b_g value.

For determining the cutoff value of the standard error, we use cross-validation.

```
# Using cross validation to determine the optimal standard Error cut off value
ses <- seq(0,1,0.1) #Range of Standard Error values
rmses <- sapply(ses, function(s){
b_g <- train_set %>%
        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) %>%
        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_set %>%
        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) %>%
        .$pred

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

# Storing the optimal standard error error value i.e the one which gives lowest RMSE
s_e <- ses[which.min(rmses)]
s_e
```

```
## [1] 0
```

We see from cross-validation that the optimal standard error cut off value is 0.6.

```
# Storing the minimum RMSE value in the rmse_results dataframe
model_5_rmse <- min(rmses)

rmse_results <- bind_rows(rmse_results,
     data_frame(Method="Movie + User + Time + Genre Effects with se Cutoff",
     RMSE = model_5_rmse ))
```

```r
print.data.frame(rmse_results)
```

```
##                                                 Method      RMSE
## 1                                     Just the average 1.0599043
## 2                                   Movie Effect Model 0.9437429
## 3                                 Movie + User Effects 0.8659320
## 4                   Movie + User Effects + Time effect 0.8654403
## 5                  Movie + User + Time + Genre Effects 0.8654403
## 6 Movie + User + Time + Genre Effects with se Cutoff 0.8654403
```

We also see that when we use a Standard Error cutoff of 0.6, the RMSE reduces to 0.8651031

3.6 Regularisation

We can see that the top 10 best movies based on the b_i are relatively obscure with very few ratings (n)

title	b_i	n
NA	1.487518	1
NA	1.487518	3
NA	1.487518	2
NA	1.487518	1
NA	1.487518	1
NA	1.487518	1
NA	1.487518	1
NA	1.487518	1
NA	1.487518	1
NA	1.404184	6

We can also see that the top 10 worst movies based on the b_i are relatively obscure with very few ratings (n)

title	b_i	n
NA	-3.012482	1
NA	-3.012482	1
NA	-3.012482	2
NA	-2.749982	40
NA	-2.667244	168
NA	-2.637482	4
NA	-2.637482	28
NA	-2.603391	11
NA	-2.512482	1
NA	-2.512482	1

Also, the top 10 users with the lowest and highest b_u, have rated relatively few movies (n).

userId	b_u	n
13496	-3.420602	15
48146	-3.239966	20
49862	-3.237758	15
6322	-3.110302	13
63381	-2.975351	13
62815	-2.965419	13
6907	-2.736872	18
15515	-2.648920	27
43628	-2.574215	16
42019	-2.550563	20

userId	b_u	n
13524	1.877483	19
18591	1.851187	15
1943	1.815695	15
56965	1.780003	22
7999	1.771917	28
45895	1.741748	13
46484	1.724557	17
36022	1.705147	73
54009	1.701676	23
46262	1.695737	240

When we look at the top 10 dates with the highest and lowest b_t values, we can see that a lot of these dates have very few number of ratings.

day	b_t	n
1996-02-07	1.7291835	1
1995-01-09	1.1808726	1
1998-05-28	1.1355743	5
1996-02-29	0.8761464	3
1999-08-31	0.7775911	9
1996-02-10	0.6183620	1
1996-02-17	0.6165702	2
1998-05-08	0.6157182	27
1999-03-21	0.6053782	14
1998-05-03	0.5240565	25

day	b_t	n
1998-06-09	-0.5382441	81
1999-04-14	-0.5208188	57
1999-03-11	-0.4974341	11
1999-09-11	-0.4799580	34
2007-10-17	-0.4671913	1631
1996-03-04	-0.3912226	6
1998-03-20	-0.3909308	22
1998-09-20	-0.3588280	264
1999-06-24	-0.3582838	20
1997-09-11	-0.3553675	9

When we look at the top 10 genres with the highest and lowest b_g values, it can again be observed that some of these genres have very few ratings

genres	b_g	n
NA	0	7200043

genres	b_g	n
NA	0	7200043

Since large errors can increase the RMSE, it is be better to be conservative when unsure.

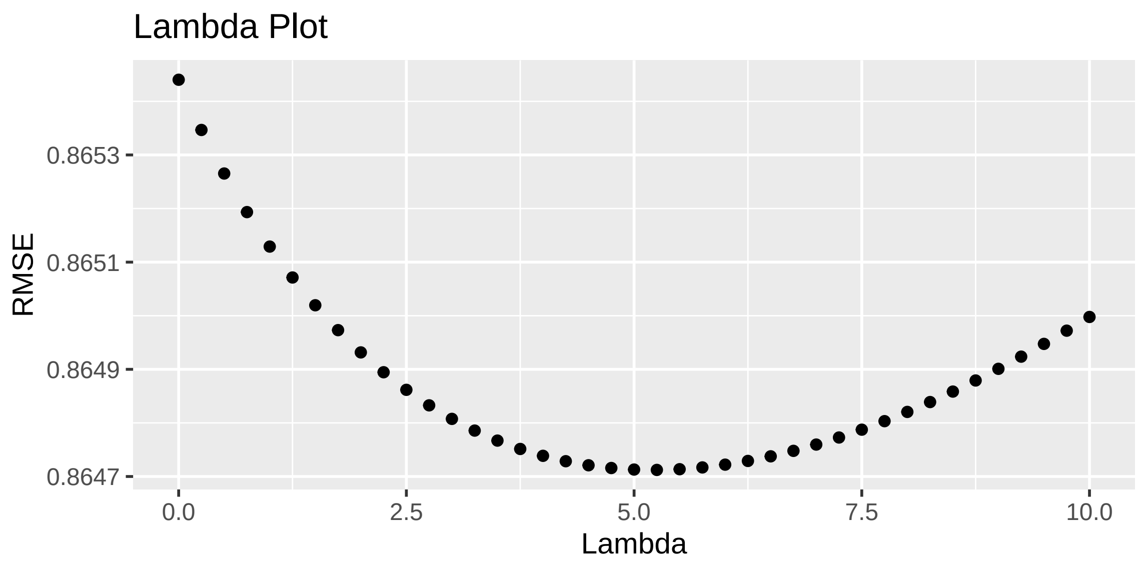
Regularization can be used to constrain the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. Instead of minimizing the least square equation, we minimize an equation that adds a penalty.

$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u-b_t-b_g\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_t b_t^2+\sum_g b_g^2\right)$

where $t = t_{u,i}$ and $g = g_{u,i}$

Since λ is a tuning parameter, cross-validation can be used to choose its optimal value.

```r
#performing cross-validation using different lambda values
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
mu <- mean(train_set$rating)

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

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

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

b_g<-train_set %>%
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) %>%
summarise(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_set %>%
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) %>%
.$pred
RMSE(predicted_ratings, test_set$rating)
return(RMSE(predicted_ratings, test_set$rating))
})

#Plotting Lambda values vs RMSE
ggplot(data.frame(Lambda=lambdas,RMSE=rmses),aes(Lambda,RMSE)) +
  geom_point() +
  ggtitle("Lambda Plot")
```



We find that the optimal value of lambda is 5.25

```r
# Storing the optimal Lambda value i.e the one which gives lowest RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.25
```

After regularisation, we also find that the RMSE value on our test set predictions has reduced substantially to 0.8643941

```r
# Storing the minimum RMSE value in the rmse_results dataframe
model_6_rmse <- min(rmses)
rmse_results <- bind_rows(rmse_results,
      data_frame(Method="Reglarised Movie + User + Time + Genre Effects",
      RMSE = model_6_rmse ))
print.data.frame(rmse_results)
```

```
##                                              Method      RMSE
## 1                                  Just the average 1.0599043
## 2                                Movie Effect Model 0.9437429
## 3                              Movie + User Effects 0.8659320
## 4                Movie + User Effects + Time effect 0.8654403
## 5               Movie + User + Time + Genre Effects 0.8654403
## 6 Movie + User + Time + Genre Effects with se Cutoff 0.8654403
## 7     Reglarised Movie + User + Time + Genre Effects 0.8647123
```

4 Results

We have finalised our model to minimize this equation:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u-b_t-b_g\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_t b_t^2+\sum_g b_g^2\right)$$

where $t = t_{u,i}$ and $g = g_{u,i}$

with tuning paramter $\lambda = 5.25$

We can now use this model and the tuning parameters to train using the entire edx dataset and then make predictions on the validation set.

```
# Training final model using edx dataset
mu <- mean(edx$rating) # Calculating mean rating in edx set

# Computing b_i for each movie as mean of residuals
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

# Computing b_u for each user as mean of residuals
b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Computing b_t for each day as mean of residuals
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")) %>%
    summarise(b_t=sum(rating-b_i-b_u-mu)/(n()+lambda))

# Computing b_g for each genre as mean of residuals
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(b_t,0)) %>% #Replacing NA values with 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_e)

# Predicitng the ratings on the validation set
predicted_ratings <- validation %>%
    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)) %>% #Replacing NA values with 0
    left_join(b_g, by="genres") %>%
    mutate(b_g=replace_na(b_g,0)) %>% #Replacing NA values with 0
    mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
    .$pred

# Calculating RMSE for predictions made on validation set
RMSE_final <- RMSE(predicted_ratings, validation$rating)
RMSE_final
```

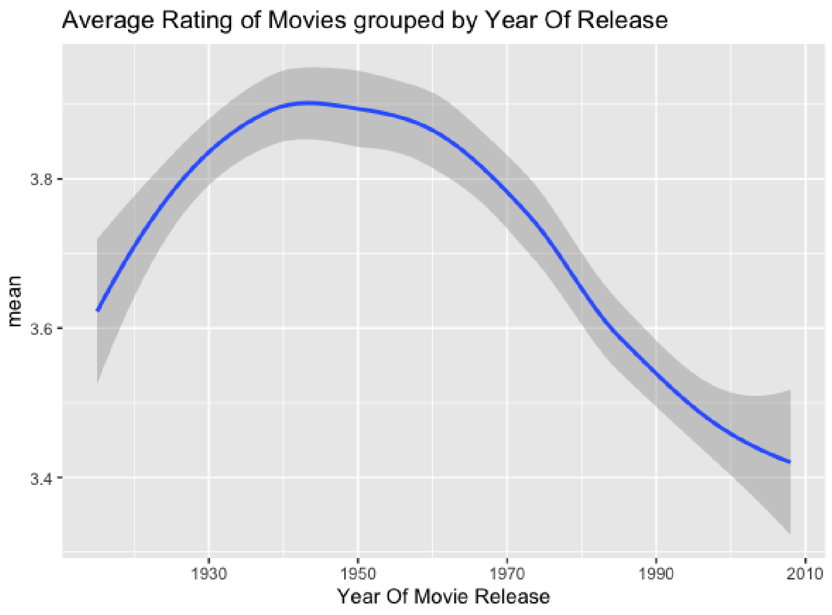
```
## [1] 0.8642738
```

We see that the RMSE on the final validation set is 0.8642738

4.1 Conclusion

In developing the MovieLens Recommendation system, we implemented a model that integrates Movie, User, Time, and Genre Effects with regularization to enhance final predictions. This model stands out for its speed, computational efficiency, and scalability.

The plot below illustrates the influence of the movie release year on ratings, suggesting a subtle effect. Although incorporating a least squares estimate of the 'Year of movie release' effect could potentially enhance RMSE, this would come at the cost of increased computation time.



It's worth noting that the estimates of time and genre effects are not user and movie-specific, but rather generalized across all users and movies. While this generalization is a pragmatic approach, conducting a user and movie-specific analysis could significantly reduce RMSE. However, such an analysis would be computationally intensive given the substantial number of users (approximately 70,000) and movies (over 10,000).

Alternative methods, such as Principal Component Analysis and Singular Value Decomposition, may offer more accurate results. However, due to the extensive size of the dataset (over 10 million entries), these analyses would be excessively computationally demanding. The chosen approach balances accuracy and computational efficiency, particularly considering the need for code to run on various computers for peer grading.