HarvardX Data Science Capstone: Movielens Project

Stephanie Phoa

7/28/2021

Contents

1 Executive Summary

This Capstone: MovieLens Project is initiated as part of the fulfilment requirements for the Data Science Professional Certificate Course by HarvardX and hosted on edX.org. The objective for this project is to attempt to model a recommender system that predicts movie ratings by users. In accordance to the Capstone MovieLens Grading Rubric, the accuracy of the prediction model will be evaluated by the Root Meaned Squared Error (RMSE) of the prediction and observed values. A RMSE of lower than 0.86490 is needed to obtain full marks under the RMSE portion of the grading rubric. Therefore, it is the goal of this project to train a prediction model that can achieve an RMSE score of under 0.86490.

2 Methodology

This project is inspired by the Netflix Challenge conducted from 2008 to 2009. This competition worth $1 million US dollars helped achieve great advancements in the machine learning and recommender systems industries.One of it's biggest achievements is the popularization of Matrix Factorization methods of training recommender algorithms. The funkSVD model designed by Simon Funk for the challenge allowed systems to use sparse user-item matrix data effectively to generate predictions.

We start this project with a raw MovieLens 10M dataset, which will go through initial preprocessing using the provided assignment code. This code splits the dataset into edX and validation data. To avoid overtraining, we will again partition the edX data into train and test sets, and reserve our validation data for the final RMSE analysis. Next, we perform Exploratory Data Analysis on the 10M MovieLens data by visualizing the dataset. Following that, we shall attempt to train a linear regression model based on movie, user, time and genre biases. In addition, we will perform regularization on the aforementioned regression model to achieve better RMSE results. We shall also train a model using the matrix factorization method on sparse data. To do this, we use the recosystem package in R. Finally, we will obtain and compare the RMSE results from all trained models. The models with RMSEs lower than 0.86490 will then be used on predict data on the validation set. Lastly we will calculate the final RMSEs using data that was trained on the validation set. We will discuss any findings and/or limitations of this project in the concluding statement.

2.1 Data Wrangling and Preprocessing

We start by downloading and installing the required packages for this project.

```
library(knitr)
library(lubridate)
library(recosystem)
library(matrixStats)
library(stringr)
library(tidyverse)
library(caret)
library(data.table)
library(tinytex)
library(kableExtra)
```

Initial data wrangling and preprocessing is done through provided code by HarvardX, which downloads and combines the raw Movielens 10M dataset and then partitioning it into *edx* and *validation* sets.

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip


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")

str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)


# if using R 4.0 or later:
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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
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 validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

We further partition the edX dataset into train and test sets to avoid overtraining on the validation set, to reduce training times, we opted to split the train and test sets with a 90:10 split.

```
# Splitting edx into train and test sets

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

test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

removed <- anti_join(temp, test)
train <- rbind(train, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Here's a preview of our freshly partitioned train ...

```
head(train)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     292      5 838983421               Outbreak (1995)
## 3:      1     316      5 838983392               Stargate (1994)
## 4:      1     329      5 838983392 Star Trek: Generations (1994)
## 5:      1     355      5 838984474       Flintstones, The (1994)
## 6:      1     356      5 838983653           Forrest Gump (1994)
##                           genres
## 1:                Comedy|Romance
## 2:  Action|Drama|Sci-Fi|Thriller
## 3:       Action|Adventure|Sci-Fi
## 4: Action|Adventure|Drama|Sci-Fi
## 5:       Children|Comedy|Fantasy
## 6:      Comedy|Drama|Romance|War
```

... and test sets

```
head(test)
```

```
##    userId movieId rating  timestamp
## 1:      1     185      5  838983525
## 2:      2     260      5  868244562
```

```
## 3:      2     590      5  868245608
## 4:      2    1049      3  868245920
## 5:      2    1210      4  868245644
## 6:      3    1148      4 1133571121
##                                                                 title
## 1:                                                  Net, The (1995)
## 2: Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
## 3:                                     Dances with Wolves (1990)
## 4:                               Ghost and the Darkness, The (1996)
## 5:            Star Wars: Episode VI - Return of the Jedi (1983)
## 6:                  Wallace & Gromit: The Wrong Trousers (1993)
##                             genres
## 1:            Action|Crime|Thriller
## 2:         Action|Adventure|Sci-Fi
## 3:         Adventure|Drama|Western
## 4:                Action|Adventure
## 5:         Action|Adventure|Sci-Fi
## 6: Animation|Children|Comedy|Crime
```

\newpage

2.2 Creating a Genre Matrix

Apart from the provided features, we decided to further explore genre and time. To explore the Genre feature, we extracted strings of distinct Genres from its column.

```
# Creating a Movie-Genre Matrix

genres  <- train %>% select(movieId, genres) %>% group_by(movieId, genres)

genres <- separate(genres, genres, c("genre_1", "genre_2", "genre_3", "genre_4", "genre_5", "genre_6",

genre_types <- sapply(2:10, function(x){
  d <- distinct(genres[,x])
})

genre_types <- unique(unlist(genre_types))[-21]

print(genre_types)
```

```
##  [1] "Comedy"             "Action"             "Children"
##  [4] "Adventure"          "Animation"          "Drama"
##  [7] "Crime"              "Sci-Fi"             "Horror"
## [10] "Thriller"           "Film-Noir"          "Mystery"
## [13] "Western"            "Documentary"        "Romance"
## [16] "Fantasy"            "Musical"            "War"
## [19] "IMAX"               "(no genres listed)"
```

Following, we create a sparse genre matrix against MovieID.

```
genres <- sapply(1:20, function(x){
  g<-genres[2:10]==genre_types[x]
  rowSums(g, na.rm = T)
})

colnames(genres) <- genre_types
```

This is what the genre matrix looks like.

```
head(genres[1:7,1:7])
```

```
##      Comedy Action Children Adventure Animation Drama Crime
## [1,]      1      0        0         0         0     0     0
## [2,]      0      1        0         0         0     1     0
## [3,]      0      1        0         1         0     0     0
## [4,]      0      1        0         1         0     1     0
## [5,]      1      0        1         0         0     0     0
## [6,]      1      0        0         0         0     1     0
```

2.3 Decoding and Converting Time

We also wanted know if the Release Year of a movie affects it's rating we converted the Timestamp column from it's POSIXct coding and pulled just the Year variable from it.

```
time <- train  %>%   select(movieId, userId, timestamp, rating) %>%
  mutate(rating_year=year(as_datetime(timestamp))) # pulling movie release year from title string

release_year <- train %>% select(movieId, title) %>%
  mutate(release_year=str_extract(train$title, "(\\d{4})")) %>%
  select(movieId, release_year) %>%
  distinct()

time <- left_join(time, release_year, by="movieId")
```

For convenience, we keep all time variables in it's own dataframe for now.

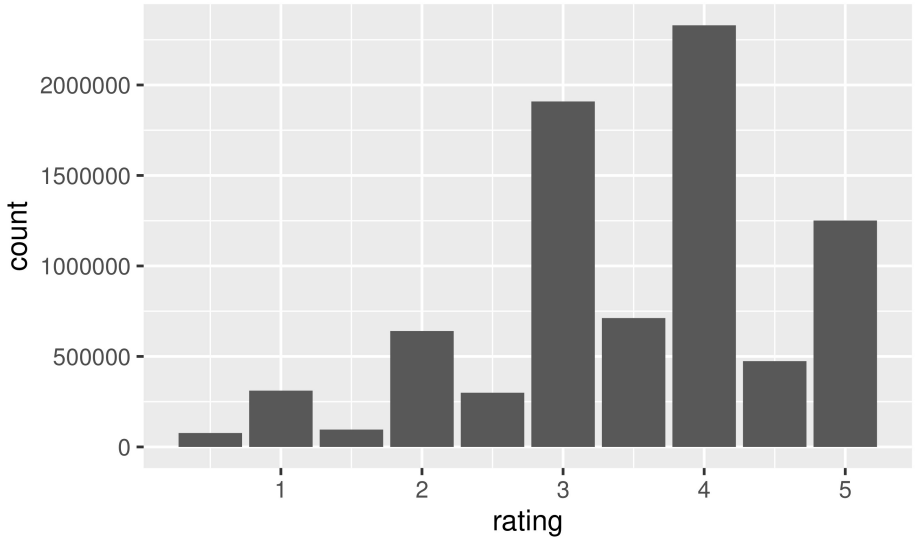
```
head(time)
```

```
##    movieId userId timestamp rating rating_year release_year
## 1:     122      1 838985046      5        1996         1992
## 2:     292      1 838983421      5        1996         1995
## 3:     316      1 838983392      5        1996         1994
## 4:     329      1 838983392      5        1996         1994
## 5:     355      1 838984474      5        1996         1994
## 6:     356      1 838983653      5        1996         1994
```

3 Exploratory Data Analysis

3.0.0.0.1 Distribution of Ratings To get a better view of our data, we performed a simple exploratory data analysis to find out what we are dealing with. First, we plotted the distribution of Ratings to identify any patterns.

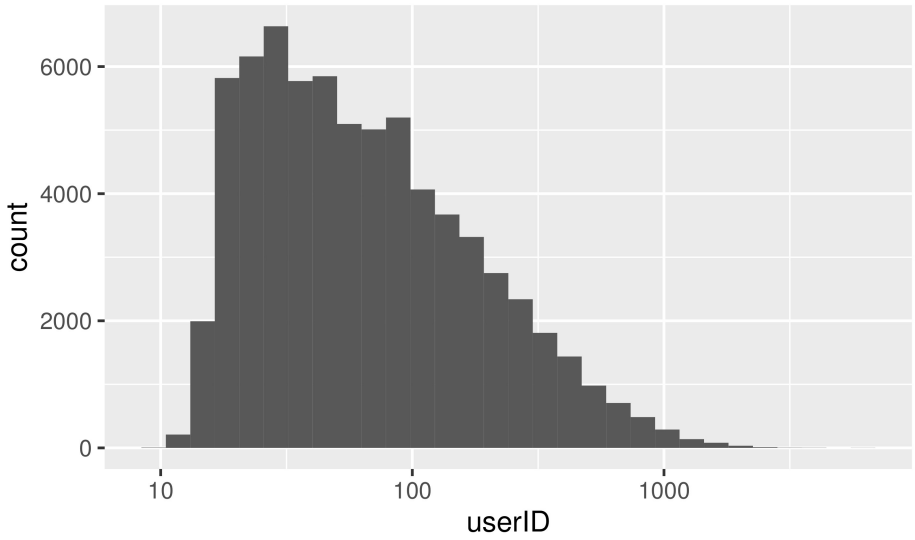
```
# ratings distribution
ggplot(train, aes(rating)) +
  geom_bar()
```



The distribution of ratings as shown in the barplot above shows a large skew towards the left. It shows most ratings tend to be higher rather than lower, and that there are many more whole number ratings (i.e: 3, 4, 5) as compared to in-between (i.e 3.5).

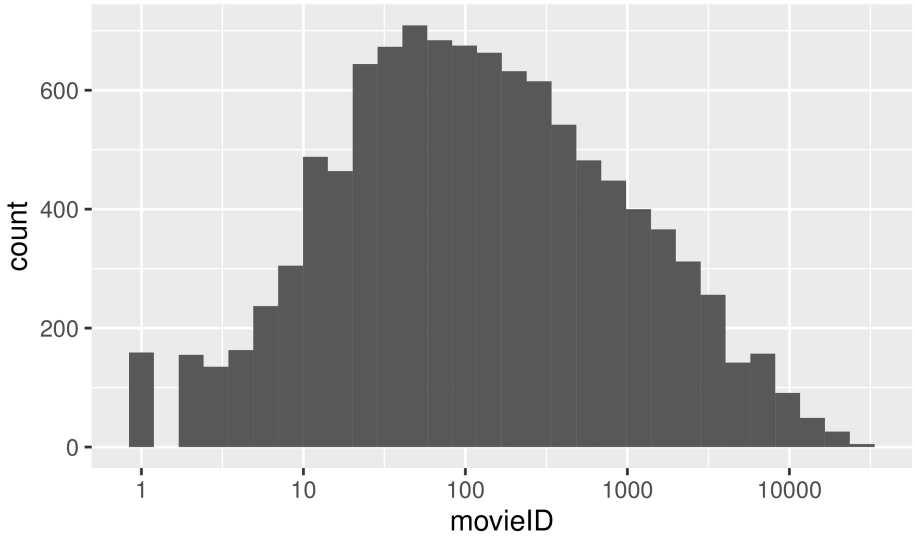
3.0.0.0.2 Distribution of Ratings by User Next we wanted to see the distribution of ratings by User ID. We expect to see older Users (with lower userIDs) to have more ratings in general.

```
# user ratings distribution
train %>% group_by(userId) %>% summarize(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  xlab("userID")
```



3.0.0.0.3 Ratings Distribution by movie To take a look at the amount of ratings each movie has received, we look at the distribution of ratings per movie.

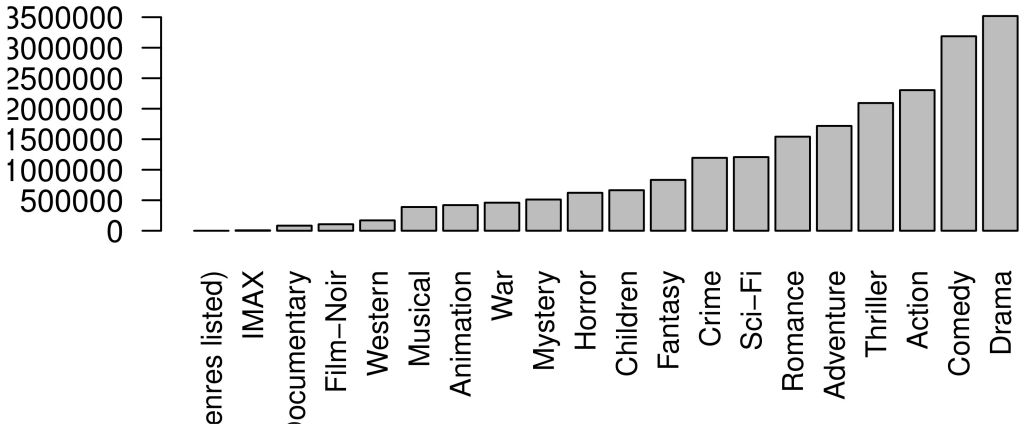
```
# movie ratings distribution
train %>% group_by(movieId) %>% summarize(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  xlab("movieID")
```




```
##            used (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2507442  134    4544781  242.8   4544781  242.8
## Vcells 49277594  376  181741845 1386.6 226250291 1726.2
```

3.0.0.0.4 Genre Prevelance Finally, we also take a look into which genres categories are more prevalent within movies.

```
# genre prevelance barplot
colSums(genres) %>% sort() %>% barplot(., las=2, cex.names=0.9)
```



4 Training: Linear Regression

Finally, we start training our models. We do our model training with our Train set on our Test set, leaving our Validation set for our final evaluation. This is so that we can avoid overfitting our models because in a real world scenario, we would not have access to our final dataset on which we will be predicting for. As per the project requirements, we will be using Root Mean Squared Error (RMSE) as our evaluation metric. We first create a RMSE function for our future evaluations.

```
RMSE <- function(trueRatings, predictedRatings){
sqrt(mean((trueRatings - predictedRatings)^2, na.rm = TRUE))}

mu <- mean(train$rating)

naive_rmse <- RMSE(test$rating, mu)
naive_rmse
```

```
## [1] 1.060054
```

4.1 Baseline Regression Model

We first start out by constructing our baseline regression model. This baseline will be used to evaluating our later models and to see if they would improve on this baseline. We use the basic linear regression formula to calculate our model, where each b represents the biases or effect of a feature in our model.

4.1.0.1 Movie Effect

The b in the regression formula represents how much a feature affects the outcome. The first feature we identified is the Movie Effect, thus named *bi*. We show our RMSE of our linear model if we only used our Movie Effect as a feature.

```
# movie effect: bi

base_bi <- train %>% group_by(movieId) %>%
  summarize(bi = mean(rating - mu))

base_yhat_bi <- test %>% left_join(base_bi, by="movieId") %>%
  mutate(yhat=mu+bi) %>% pull(yhat)

bi_rmse<- RMSE(test$rating, base_yhat_bi) #rmse bi only

bi_rmse
```

```
## [1] 0.9429615
```

4.1.0.2 User Effect

We also found out that individual users have an effect on the ratings given. This effect will account for any User biases such as a tendency to like certain genres or perhaps give harsher ratings.

```
# user effect: bu

base_bu <- train %>%
left_join(base_bi, by="movieId") %>%
```

```
group_by(userId) %>%
summarize(bu = mean(rating - mu- bi ))

base_yhat_bu <- test %>% left_join(base_bi, by="movieId") %>%
  mutate(yhat=mu+bi) %>% pull(yhat)
bu_rmse<- RMSE(test$rating, base_yhat_bu) #rmse bu only

bu_rmse
```

```
## [1] 0.9429615
```

4.1.0.3 Baseline RMSE

We combine our Movie and User Effects into our model and evaluate it with the RMSE function earlier. This will be our Baseline RMSE.

```
base_yhat <- test %>%
left_join(base_bi, by="movieId") %>%
left_join(base_bu, by="userId") %>%
mutate(yhat=mu+bi+bu) %>% pull(yhat)

baseline_rmse <- RMSE(test$rating, base_yhat) #rmse bi+bu regression

baseline_rmse
```

```
## [1] 0.8646843
```

4.2 Regularized Regression Model

4.2.0.1 Finding Lambda

To try to get a better RMSE for our models, we attempt to regularize it using a penalty term, lambda. To find lambda, we using a tune our baseline model with the penalty term, and plot it to see it's effect. The following is our tuning function.

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(x){

  base_bi <- train %>% group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+x))  %>% suppressMessages()

  base_bu <- train %>%
    left_join(base_bi, by="movieId") %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - mu- bi )/(n()+x))  %>% suppressMessages()

  base_bt <- train %>%
    left_join(base_bi, "movieId") %>%
    left_join(base_bu, "userId") %>%
    left_join(release_year, "movieId") %>%
    mutate(bt=sum(rating - mu- bi- bu)/(n()+x), n_t = n()) %>%
    group_by(release_year) %>%
    summarize(bt=mean(rating-mu-bi-bu))  %>% suppressMessages()
```

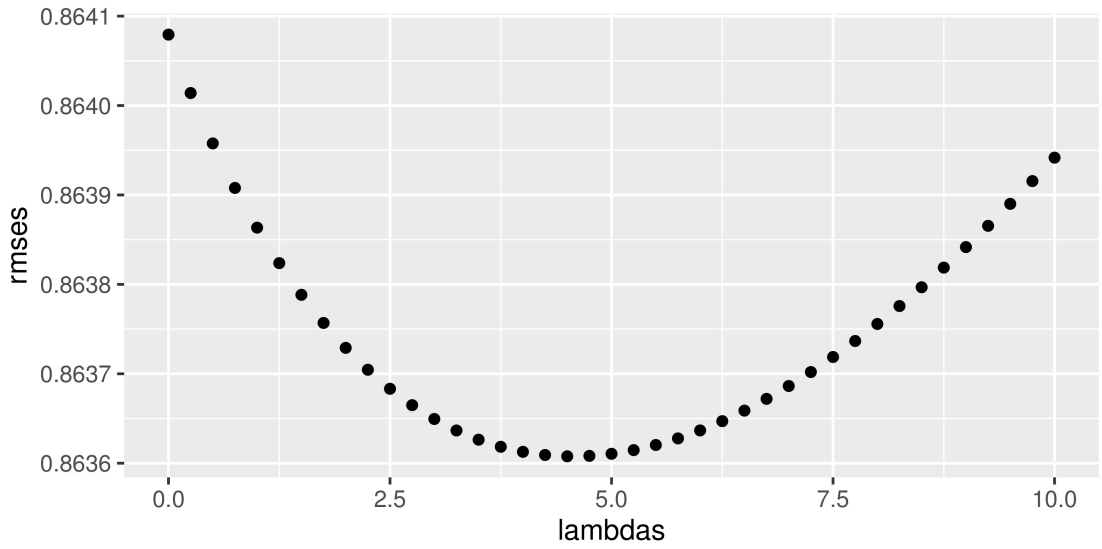
```
  base_bg <- train %>%
    left_join(release_year, by="movieId") %>%
    left_join(base_bi, "movieId") %>%
    left_join(base_bu, "userId") %>%
    left_join(base_bt, "release_year") %>%
    mutate(bg=sum(rating - mu- bi - bu - bt  )/(n()+x), n_g = n()) %>%
    group_by(genres) %>%
    summarize(bg=mean(rating-mu-bi-bu-bt))  %>% suppressMessages()

  y_hat <- test %>%
    left_join(release_year, by="movieId") %>%
    left_join(base_bi, by="movieId") %>%
    left_join(base_bu, by="userId") %>%
    left_join(base_bt, by="release_year") %>%
    left_join(base_bg, by="genres") %>%
    mutate(yhat=mu+bi+bu+bt+bg) %>% pull(yhat)  %>% suppressMessages()

  return(RMSE(test$rating, y_hat))
})
```

We plot our RMSEs and notice a clear pattern, we can easily identify our best penalty value, lambda, to be around 4.5.

```
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 4.5
```

```
lambda <- lambdas[which.min(rmses)]
```

4.2.0.2 Baseline Regression with Regularization

Knowing our lambda value, we then combine it with our initial baseline model in order to regularize it.

```
# bi regularized

train_bi <- train %>% group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(n()+lambda), n_i = n())

yhat_bi <- test %>% left_join(train_bi, by="movieId") %>%
  mutate(yhat=mu+bi) %>% pull(yhat)
reg_bi<- RMSE(test$rating, yhat_bi)

# bu regularized

train_bu <- train %>%
  left_join(train_bi, by="movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - mu- bi )/(n()+lambda), n_u = n())

yhat_bu <- test %>% left_join(train_bu, by="userId") %>%
  mutate(yhat=mu+bu) %>% pull(yhat)
reg_bu <- RMSE(test$rating, yhat_bu)

# bi + bu regularized

yhat_bi_bu <- test %>%
  left_join(train_bi, by="movieId") %>%
  left_join(train_bu, by="userId") %>%
  mutate(yhat=mu+bi+bu) %>% pull(yhat)
reg_bi_bu <- RMSE(test$rating, yhat_bi_bu)
```

We find that our RMSE has improved slightly over the baseline model with the addition of regularization.

```
print(reg_bi_bu)
```

```
## [1] 0.8641391
```

4.3 Genre and Time Effects

During our exploration of the data, we found there were significant effects on ratings by Genre and Time. So in an attempt to improve our model, we included these new effects into our regularized regression model. We start by including the Time Effect (bt).

```
# time effect: bt
# mu + bi + bu + bt

train_bt <- train %>%
  left_join(train_bi, "movieId") %>%
  left_join(train_bu, "userId") %>%
  left_join(release_year, "movieId") %>%
  mutate(bt=sum(rating - mu- bi- bu)/(n()+lambda), n_t = n()) %>% # with regularization
  group_by(release_year) %>%
```

```
  summarize(bt=mean(rating-mu-bi-bu))

yhat_bt <-test %>%
  left_join(release_year, by="movieId") %>%
  left_join(train_bi, by="movieId") %>%
  left_join(train_bu, by="userId") %>%
  left_join(train_bt, by="release_year") %>%
  mutate(yhat=mu+bi+bu+bt)%>% pull(yhat)

reg_bi_bu_bt  <- RMSE(test$rating, yhat_bt)
reg_bi_bu_bt
```

```
## [1] 0.8638357
```

We can see our RMSE score with just time effects have lowered significantly, we hope to see further improvement with our Genre score.

```
# genre effect: bg
# mu + bi + bu + bt + bg

train_bg <- train %>%
  left_join(release_year, by="movieId") %>%
  left_join(train_bi, "movieId") %>%
  left_join(train_bu, "userId") %>%
  left_join(train_bt, "release_year") %>%
  mutate(bg=sum(rating - mu- bi - bu - bt  )/(n()+lambda), n_g = n()) %>%  # with regularization
  group_by(genres) %>%
  summarize(bg=mean(rating-mu-bi-bu-bt))


yhat_bg <-test %>%
  left_join(release_year, by="movieId") %>%
  left_join(train_bi, by="movieId") %>%
  left_join(train_bu, by="userId") %>%
  left_join(train_bt, by="release_year") %>%
  left_join(train_bg, by="genres") %>%
  mutate(yhat=mu+bi+bu+bt+bg)%>% pull(yhat)

reg_bi_bu_bt_bg <- RMSE(test$rating, yhat_bg)
reg_bi_bu_bt_bg
```

```
## [1] 0.8636077
```

With our Genre Effects, our regression model's RMSE has dropped below the expected RMSE for the project course. Still, we would like to explore an alternative model to see if we can get a better RMSE score.

To sum it up, here is a table of RMSEs for each step in our linear regression model.

Model	RMSE	Reg_RMSE
Naive - Just the Average	1.06005370222409	NA
Movie Effect	0.942961498004501	0.942967628823318
User Effect	0.942961498004501	0.993034153913614
Movie+User Effect	0.86468429490229	0.863835728387008
Movie+User+Time+Genre Effect	NA	0.863607732947358

5 Training: Matrix Factorization

Because not all Users rated all Movies, we know that if we created a User-Movie matrix, there will be a lot of blank entries. This is otherwise known as a Sparse Matrix. A sparse matrix poses many challenges to machine learning predictions as a machine learning algorithm cannot train on no data. A common mistake is to fill the gaps in datas with zeros, which would throw off the algorithm because although a user did not rate a certain movie, it doesn't mean that they disliked the movie, as a rating of zero would imply. The matrix factorization method attempts to fill in the gaps in data with a predicted score based on known features. The biggest downside of matrix factorization in machine learning is that it takes a very long time to process large amounts of data. That is why for this project, we will be using the recosystem package as it allows us to perform matrix factorization on this large set of data for a fraction of the time thanks to it's parallel computing capabilities. We note that this process still takes about 30 minutes to finish.

```
library(recosystem)

r <- Reco()

train_mf <- train %>% select(userId,movieId, rating) %>% as.matrix()
test_mf <- test %>% select(userId,movieId, rating) %>% as.matrix()

write.table(train_mf, file="train_mf.txt", sep=" ", row.names=F, col.names=F)
write.table(test_mf, file="test_mf.txt", sep=" ", row.names=F, col.names=F)

set.seed(1, sample.kind="Rounding")
train_mf<- data_file("train_mf.txt", index1=TRUE)
test_mf<- data_file("test_mf.txt", index1=TRUE)

# tuning parameters
tune <- r$tune(train_mf, opts=list(dim=c(10,20,30), lrate=c(0.1,0.2), costp_l1=0, costq_l1=0, nthread=1
tune$min

# training and prediction

set.seed(1, sample.kind="Rounding")
fit_mf <- suppressWarnings(r$train(train_mf, opts= c(tune$min,nthread=1, niter=20)))

pred_file <- tempfile()
r$predict(test_mf, out_file(pred_file))

yhat_mf<- print(scan(pred_file))

# RMSE matrix factorization
mf_rmse <- RMSE(test$rating, yhat_mf)
mf_rmse
```

As shown below, the RMSE score for the Matrix Factorization model seems to be much better than linear regression. Therefore for our final training on our Validation set, we will be using our Matrix Factorization model.

```
# RMSE matrix factorization
mf_rmse <- RMSE(test$rating, yhat_mf)
mf_rmse
```

```
## [1] 0.7859549
```

6 Final Training on Validation Set

From training our model on the Train and Test set, we have found that both regularized linear regression model, and the matrix factorization model achieve an RMSE lower than 0.864, which would receive full marks for this project. However if we compare both models, it is very clear the matrix factorization model worked much better. Therefore for our final validation set, we shall only train it on the matrix factorization model.

```
library(recosystem)

r <- Reco()

train_mf <- train %>% select(userId,movieId, rating) %>% as.matrix()
valid_mf <- validation  %>% select(userId,movieId, rating) %>% as.matrix()

write.table(train_mf, file="train_mf.txt", sep=" ", row.names=F, col.names=F)
write.table(valid_mf, file="valid_mf.txt", sep=" ", row.names=F, col.names=F)

set.seed(1, sample.kind="Rounding")
train_mf<- data_file("train_mf.txt", index1=TRUE)
valid_mf<- data_file("valid_mf.txt", index1=TRUE)

# tuning parameters
tune <- r$tune(train_mf, opts=list(dim=c(10,20,30), lrate=c(0.1,0.2), costp_l1=0, costq_l1=0, nthread=1
tune$min

# training and prediction

set.seed(1, sample.kind="Rounding")
fit_mf <- suppressWarnings(r$train(train_mf, opts= c(tune$min,nthread=1, niter=20)))

pred_file <- tempfile()
r$predict(valid_mf, out_file(pred_file))

yhat_mf_final<- print(scan(pred_file))

# RMSE matrix factorization
mf_rmse_final <- RMSE(validation$rating, yhat_mf_final)
```

The final RMSE achieved is as follows:

```
mf_rmse_final
```

```
## [1] 0.7863249
```

7 Conclusion and Limitations

In this machine learning project assigned by HarvardX's Data Science Capstone course on edX, our main objective was to predict movie ratings given a dataset with multiple features. Two distinct models were tested: the first a linear regression model with regularization, the second a matrix factorization model. Both models we tested achieved the goal RMSE score of less than 0.864, but overall, the matrix factorization

model achieved a lower score. Therefore, the latter model was used to train on our final dataset. Our final RMSE achieved is 0.786.

The biggest limitation of this project is the size of the dataset. This made a lot of algorithms quite difficult to train on because it would have taken too long. These models were chosen because of it's fast and parallel computing speed. Perhaps if we were able to train this dataset on other models, we might be able to achieve a greater RMSE. Technological limitations such as computer hardware or programming language limitations also count towards this issue. Without these limitations, it would be interesting for this dataset to be explored further to hopefully achieve a better RMSE score.