HarvardX PH125.9x Data Science: Capstone Project
Recommendation System Using the MovieLens 10M Dataset

Luke Holmes Ph.D., University of Essex

2022-11-02

Introduction

Background

The MovieLens project (https://movielens.org/) was founded in 1997 by the GroupLens research group at the University of Minnesota, and funded by the National Science Foundation. The original purpose of the research project was to explore personalised recommendation systems which, in 1997, were in their infancy. However, since then, the dataset has seen use in a wide range of applications; it is commonly used as a project for people training in Machine Learning, but it has also been used extensively in research, and even the social behaviour of its users have been studied by Psychologists.

Dataset

Broadly speaking, the MovieLens dataset is a large collection of information about movies. Each movie is associated with a wide range of attributes: These include basic information such as Title, Release Year and Genre, as well as more detailed information such as Director, Cast, DVD Release Date, and User-Created Tags. Alongside this information, each movie can be rated by any user who registers up on the website. These ratings range from 0.5 stars (the lowest rating) to 5 stars (the highest rating), and users are free to rate as many or as few movies as they wish.

By combining the information on a movie's attributes and information on a user's ratings, the overall "goal" of the website is to provide users with recommendations for movies that they have not yet seen, but may enjoy. These predictions can be formed in a number of ways: For example, by recommending a user to watch movies which are similar to ones they have previously enjoyed, or by recommending movies enjoyed by users who are deemed similar to the target user. The most recent version of the MovieLens dataset has 20 million ratings over 27,000 movies by 138,000 users. However, to save on time and computing power, the current project will use only the 10M version of the data set.

Project Goal

The current project is a capstone coursework project for the HarvardX PH125.9x Data Science course. Our goal is to predict, as accurately as possible, ratings that users will give to movies. To do this, we will train a series of Machine Learning models to predict these ratings using the other information included in the data. The dataset will be split into training and testing subsets, and the overall performance of each model will be assessed by measuring the Residual Mean Squared Error of its predicted ratings against the true ratings found in the testing set. The function we will use for measuring the RMSE of each model is given by the following code:

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

Key Steps

- **Introduction**: Background, data set and project goals will be explained
- **Method**: Data will be explored using summary statistics and visualisations
- **Model Training**: A series of Machine Learning models will be trained on the training set, with the aim of minimising loss on predictions
- **Final Model Evaluation**: The most successful Machine Learning model will be tested on the final holdout test data set
- **Conclusions**: The conclusions which can be drawn from the models will be explained

Methods

Obtaining the Dataset

The dataset must first be downloaded, unzipped, and the columns renamed appropriately.

```
# Downloads, unzips, and appropriately renames columns of the MovieLens Dataset
# Note: The following may take a few minutes, since the download is 63Mb
dl <- tempfile()
download.file("https://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")
```

For the purpose of assessing the final model, the data will be split into two sets: The edx set is the primary training set, and will be used for the building and training of all models. The validation set is the primary test set, and will be reserved for assessing the RMSE of the final model only.

```
# Create validation set from 10% of the data:
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,]

# 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)

# Clean up environment
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Finally, since we will need to train and test models using only the edx data (and are reserving the validation dataset for the final model), we must further split the edx data itself into a training and test set. These will be named train_set and test_set, respectively. At the same time, we will also implement the function to measure the RMSE of models.

```
# splitting the edx data into training and test sets
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# using semi-join to ensure that users and movies are not in both sets
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# we will assess the performance of models by computing their Residual Mean
# Squared Error (where lower scores are better) using this function:
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

# set tibbles to display 7 digits
options(pillar.sigfig = 7)
```

Data Exploration

We will first look at the variables available in our data set.

```
## Column Names:  userId movieId rating timestamp title genres
```

In order, we have:

- **userId**: A unique ID given to each user in order of registration.
- **movieId**: A unique ID given to each movie in order of being listed on the site.
- **rating**: The rating given by the user in question to the movie in question, ranging from 0.5 to 5.0.
- **timestamp**: The time when the rating was given, given in seconds since January 1st, 1970.
- **title**: The title of the movie. Includes year of original release.
- **genres**: A concatenated list of genres to which the movie belongs.

Therefore, it is the case that each row in the dataset represents a rating given by a specific user to a specific movie. Each user and each movie therefore takes up many rows in the dataset. The data set we are dealing with is a large one:

```
## edx Height: 9000055
```

```
## edx Width: 6
```

Yet, when we look at the number of unique users and movies, we make a curious finding:

```
##   Unique_Users Unique_Movies
## 1        69878         10677
```

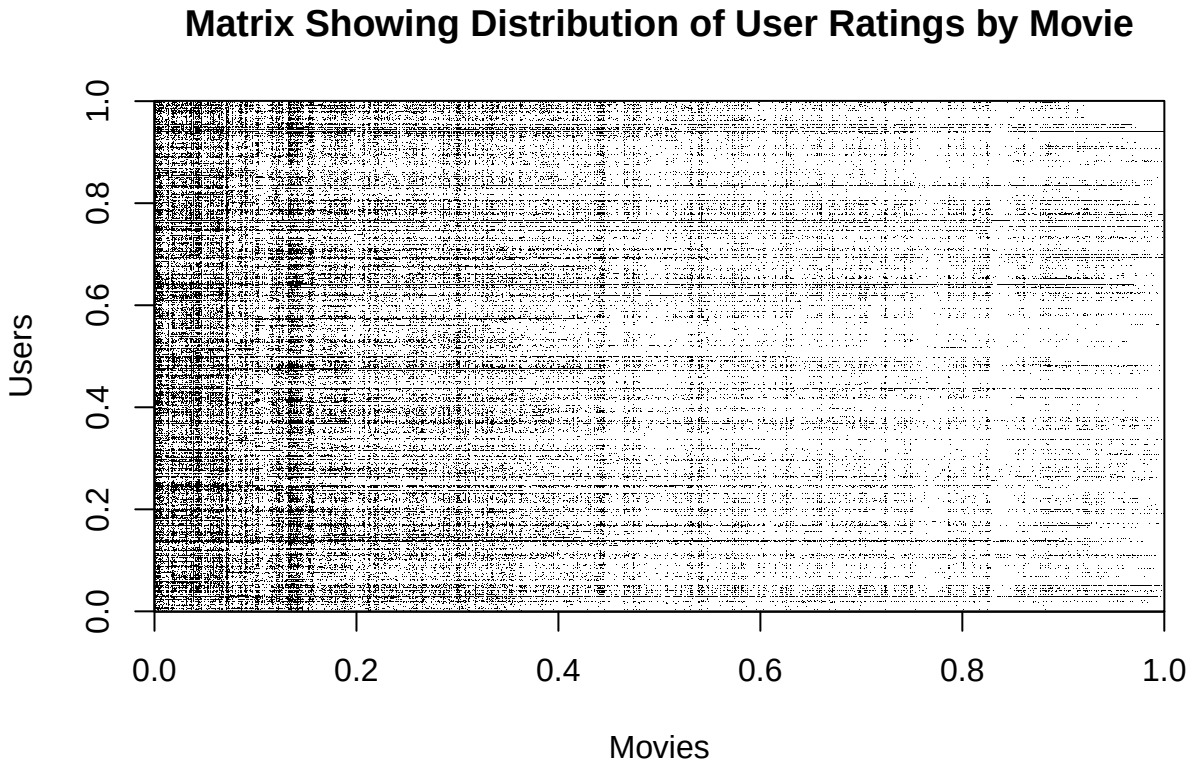
```
## Users * Movies =  746087406
```

```
## Our Row Total  =  9000055
```

If we multiply the number of unique users and unique movies together, we get a number far larger than the amount in our data set. This is because not every user has rated every movie.

Data Visualisation

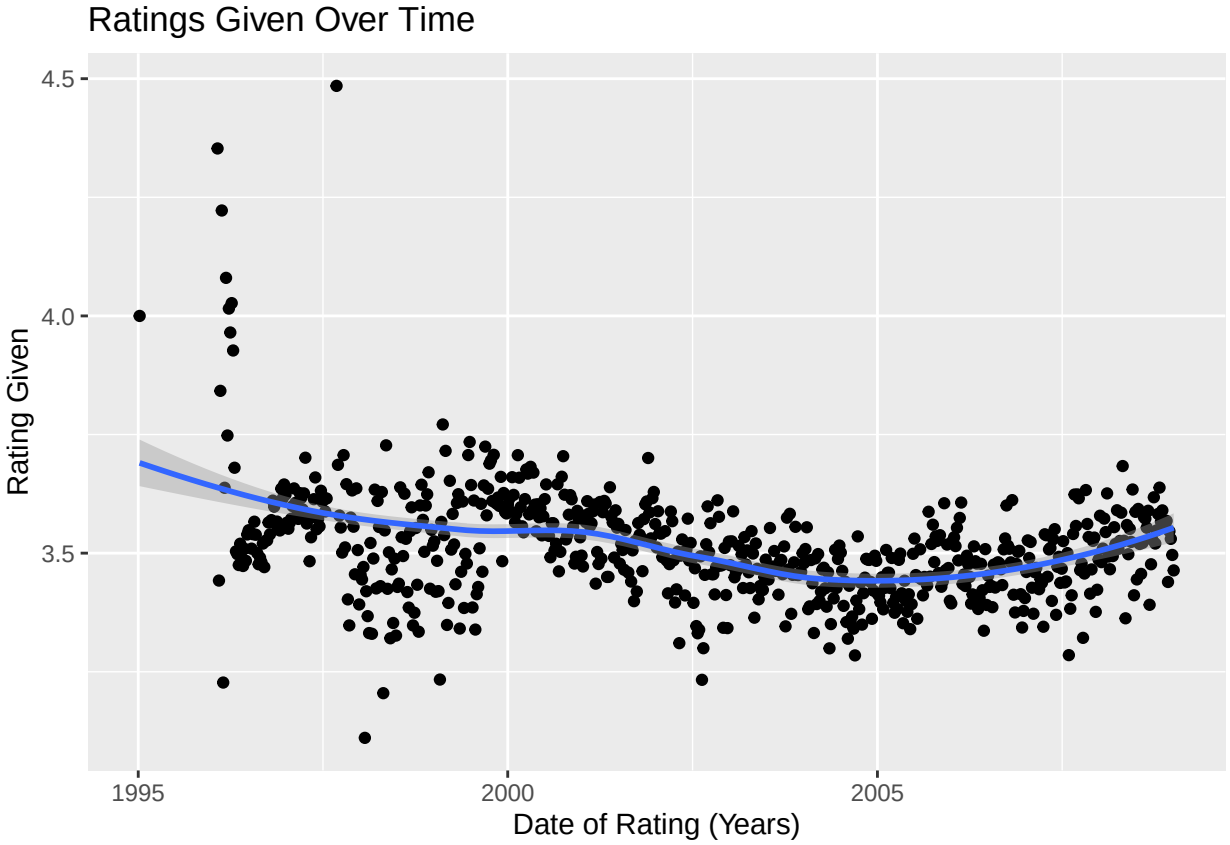
To explore this, we can create a matrix which shows which movies (x-axis) have been rated by which unique users (y-axis). On this graph, a black mark means that a given user has rated a given movie. Note that this chart covers all 10,677 movies in the dataset, but uses a random sample of 1000 users.



We can draw two main conclusions from this: Firstly, some users have given many ratings and some movies have received many ratings. This is evident from the patterns of horizontal and vertical lines visible on the

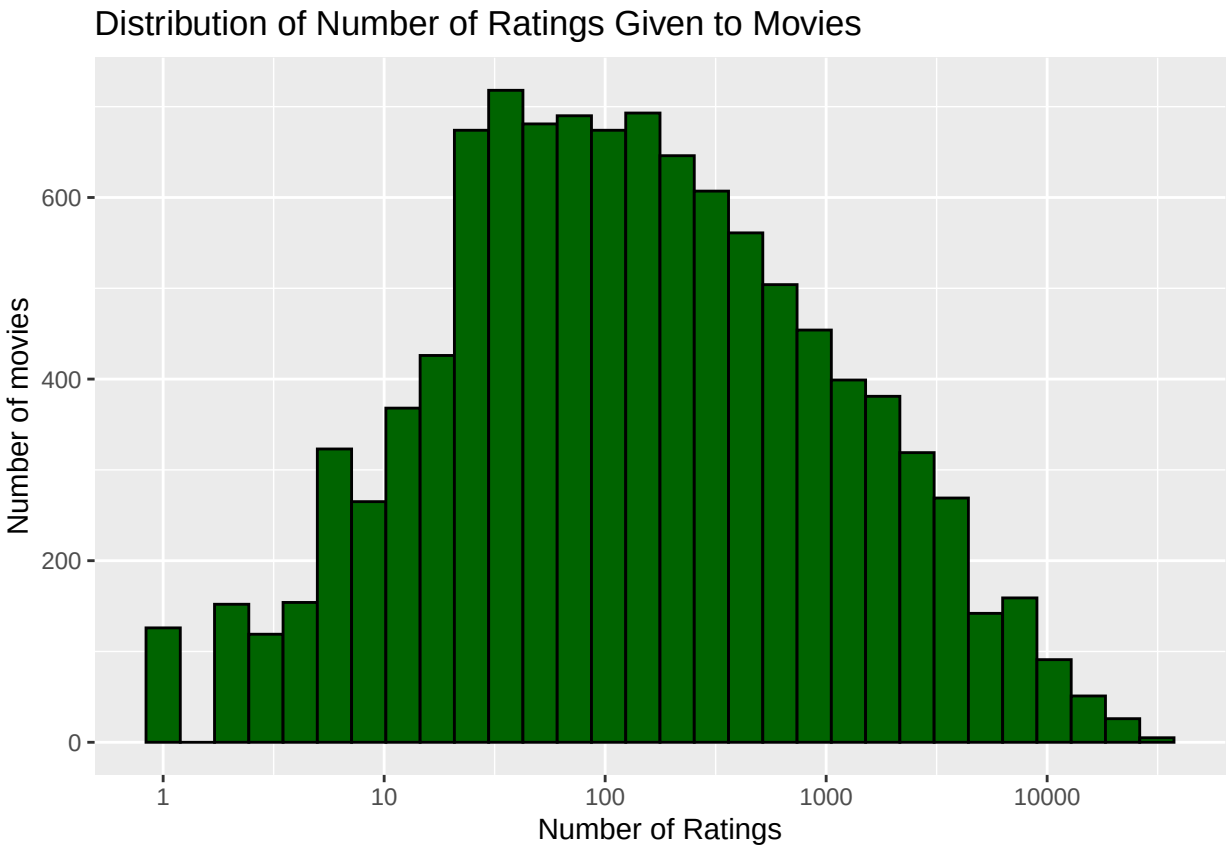
chart, and we will investigate this further. Secondly, we must note how sparse the chart is - not surprisingly, the vast majority of movies have not been rated by the vast majority of users. Thus, there is a lot of missing data in the set.

Next, we will examine the potential effect of timestamp on rating. In order to allow for more effective grouping of participants based on timestamp, we will convert the original timestamp score (given in seconds) to a more reasonable bin width of one week. This will be repeated throughout the analysis in both testing and training sets.

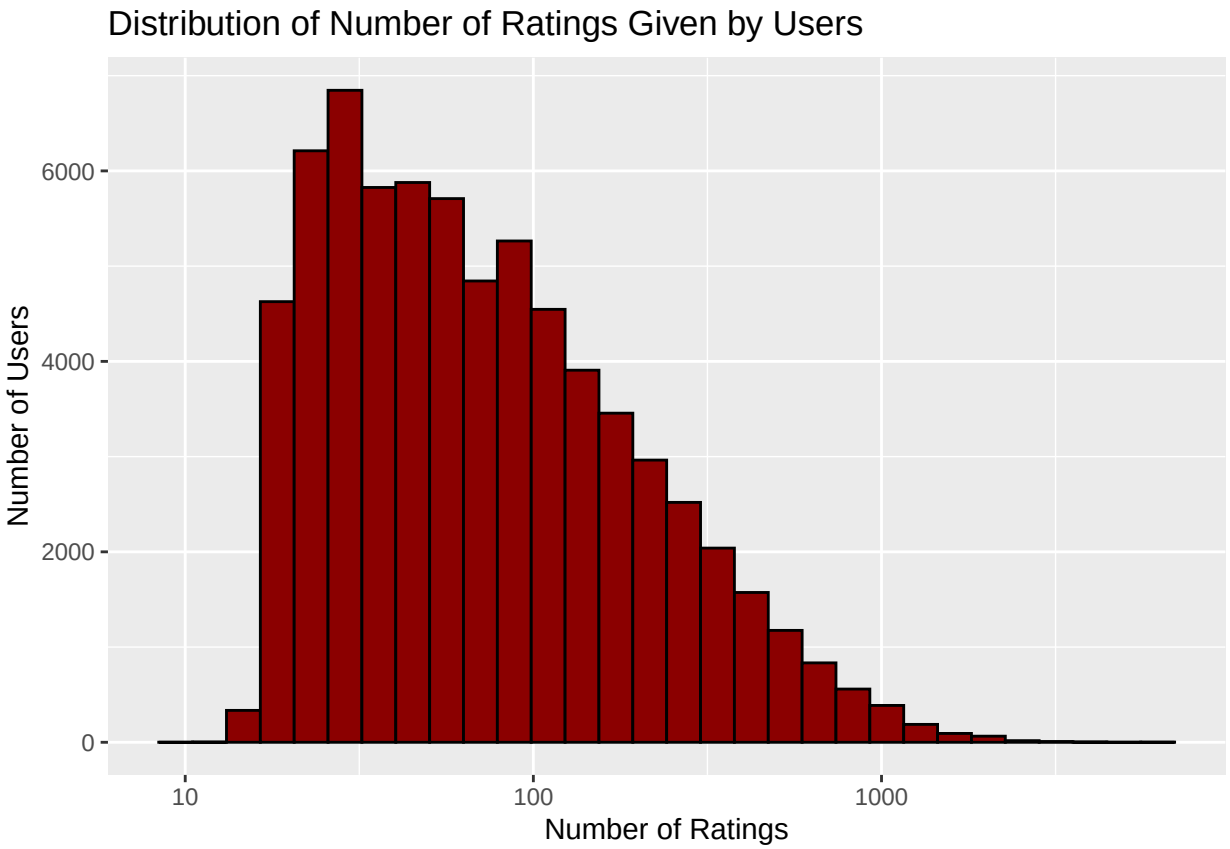


Here we can see that - perhaps against expectations - time does appear to have an effect on the rating itself. Specifically, the ratings average downwards from 1997 to around 2005, then begin gently rising again. Therefore, timestamp may be a useful predictor of ratings in our final model, and we will attempt to incorporate it.

The final step is to examine the variance in the number of ratings given by each user, and the number of ratings given to each movie. This will inform our approach to regularisation during model development. We will first look at the number of ratings received by each movie:



As expected, some movies are very popular and thus have a lot of ratings, and other movies are more obscure and have received very few ratings.



Similarly, some users are prolific and have rated many movies, and other users only used the site for a brief time and rated only a few movies. It is possible that these small sample sizes may produce unreliable results. We will therefore attempt to improve our models by accounting for this variation through regularisation.

Insights and Model Building

Exploring and visualising the data has given us several valuable insights which we must take into account when building our models. In summary, they are:

1. There are a great deal of missing ratings in the data set - most users have not rated most movies.
2. Unexpectedly, timestamp does appear to have an impact on the rating given by a user - it will therefore be incorporated into our models as a predictor.
3. There is a lot of variance in the number of ratings given to each movie and the number of ratings given by each user. In order to control for this, we will regularise our models in order to further minimise their loss.

Results

Model Training

With the information we have gathered from examining the data set, we will now build a series of Machine Learning models. We will begin with less complex models and will continually add predictors and/or tweak regularisation to try and minimise the loss of the model. After each model, we will add the RMSE of each model to a table for easy comparison.

All models will be trained and tested on the train_set and test_set datasets, which are themselves subsets of the edx dataset. Once it is clear which model gives the best performance (measured by the most minimised RMSE), we will train this final model on the full edx data set and then use it to predict the validation set for the final result.

Model 1: Naive Bayes Prediction (Midpoint)

This first model is the simplest one possible: We will predict movie ratings using the midpoint of the scale, which is 2.75 (because the scale runs from 0.5 to 5.0)

```
# predicts scores using the midpoint value of 2.75
model1 <- rep(2.75, nrow(train_set))
model1_rmse <- RMSE(test_set$rating, model1)
```

method	RMSE
Model 1: Scale Midpoint	1.306064

This model produces a very poor RMSE. Simply using the midpoint of the scale is not a good predictor, given the amount of variance in ratings between movies. However, it serves as a useful baseline to compare our future models to.

Model 2: Naive Bayes Prediction (Mean)

The second model uses the mean rating across all movies (not the mean rating for each individual movie) as a predictor of future ratings.

```
model2 <- mean(train_set$rating)
model2_rmse <- RMSE(test_set$rating, model2)
```

method	RMSE
Model 1: Scale Midpoint	1.306064
Model 2: Mean of All movies	1.060704

This model provides a better RMSE than the first model, but there is still significant room for improvement if we tailor our prediction to each individual movie.

Model 3: Average of Each Movie

The third model uses the mean rating of each individual movie to predict its future ratings.

```
# predicts scores using the mean score of each movie
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  # calculates difference between each movie mean and the overall mean
  summarize(b_i = mean(rating - mu))
# trains model 3
```

```
model3 <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
model3_rmse <- RMSE(model3, test_set$rating)
```

method	RMSE
Model 1: Scale Midpoint	1.3060641
Model 2: Mean of All movies	1.0607045
Model 3: Mean of Each Movie	0.9437144

This model causes a significant improvement in the overall RMSE.

Model 4: Average of Each User & Movie

The fourth model adds individual user score means as a predictor, in addition to the individual movie score means used in model 3.

```
# predicts scores using the mean score of each movie + each user
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  # calculates difference between each user mean and the overall mean
  summarize(b_u = mean(rating - mu - b_i))
# trains model 4
model4 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model4_rmse <- RMSE(test_set$rating, model4)
```

method	RMSE
Model 1: Scale Midpoint	1.3060641
Model 2: Mean of All movies	1.0607045
Model 3: Mean of Each Movie	0.9437144
Model 4: Movie & User Means	0.8661625

Again, the RMSE is significantly improved. By using the means of both movies and users, we can make the most accurate predictions yet.

Model 5: Average of Each User & Movie with Time

As we found during the exploratory analysis, it is possible that the date a rating was given might impact on the rating itself, with a downwards trend from 1997 to 2005 and and an upwards trend afterwards. Prior to building the model, to allow for easier grouping, we again convert timestamp from seconds into week-long bins in both training and test sets. We also continue to include individual movie and user rating means as predictors in this model.

```
# timestamp is first converted into week-long bins to be more useful for grouping
train_set <- train_set %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week"))
# the same must be done in the test set to allow joining
test_set <- test_set %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week"))
# predicts scores using the mean score of each movie + each user + timestamp
time <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(week) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
# trains model 5
model5 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(time, by='week') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  .$pred
model5_rmse <- RMSE(test_set$rating, model5)
```

method	RMSE
Model 1: Scale Midpoint	1.3060641
Model 2: Mean of All movies	1.0607045
Model 3: Mean of Each Movie	0.9437144
Model 4: Movie & User Means	0.8661625
Model 5: Movie & User Means with Time	0.8660880

This provides an improvement of RMSE, although only a marginal one. It is possible that the variance explained by timestamp is relatively minor compared to the user and movie means.

Model 6: Average of Each User & Movie with Regularisation

During the exploratory analysis, we also found that there is a great deal of variance in the number of ratings given by each user, and the numbers of ratings given to each movie. This creates a problem for us when we attempt to build models - essentially, when a movie or user has a low number of ratings, their small sample size can produce unreliable results. We will therefore attempt to improve our model further by regularising these results, which will reduce the impact of these unreliably small samples on the overall outcome of the model.

Because the RMSE improvement of adding Timestamp as a predictor was so marginal, we will first attempt to regularise the User + Movie model (4), then repeat the process on the User + Movie + Time model (5) in order to select the one with the best results.

```
# selecting the optimal lambda (and thus minimal RMSE) for User & Movie model
# Note: This code takes a long time to run
lambdas <- seq(0, 10, 0.25)
rmses6 <- 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))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
# optional: display lambda optimisation graph and minimised lambda (4.75)
# lambdas[which.min(rmses6)]
# qplot(lambdas, rmses6)
# corresponding minimised RMSE is saved
model6_rmse <- min(rmses6)
```

method	RMSE
Model 1: Scale Midpoint	1.3060641
Model 2: Mean of All movies	1.0607045
Model 3: Mean of Each Movie	0.9437144
Model 4: Movie & User Means	0.8661625
Model 5: Movie & User Means with Time	0.8660880
Model 6: Regularised Movie & User Means	0.8655425

The regularisation process has yielded a marginal improvement in the RMSE.

Model 7: Average of Each User, Movie & Time with Regularisation

Finally, we will apply regularisation to the User + Movie + Time model (5), to see if this further improves the RMSE.

```
# selecting the optimal lambda (and thus minimal RMSE) for User, Movie & Time model
# Note: This code takes an even longer time to run
lambdas <- seq(0, 10, 0.25)
rmses7 <- 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_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(week) %>%
    summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+l))
```

```
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "week") %>%
    mutate(pred = mu + b_i + b_u + b_t) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
# optional: display lambda optimisation graph and minimised lambda (5)
# lambdas[which.min(rmses7)]
# qplot(lambdas, rmses7)
# corresponding minimised RMSE is saved
model7_rmse <- min(rmses7)
# knits the RMSE of model 7 to a table
```

method	RMSE
Model 1: Scale Midpoint	1.3060641
Model 2: Mean of All movies	1.0607045
Model 3: Mean of Each Movie	0.9437144
Model 4: Movie & User Means	0.8661625
Model 5: Movie & User Means with Time	0.8660880
Model 6: Regularised Movie & User Means	0.8655425
Model 7: Regularised Movie & User Means with Time	0.8654348

Again, the RMSE is marginally improved, making this model (Regularised Movie & User Means with Time) the most effective one. This is therefore the model that we will select to apply to the validation data set.

Final Model Evaluation

Now that the most effective model has been determined, we are ready to evaluate it against the full validation data set. For the sake of computational power, we will begin by clearing the entire workspace except for the edx and validation datasets. We will then re-train the model on the full edx set (9,000,000 items) rather than the train_set subset (7,200,000 items) before performing the final evaluation. This will allow us to make the best possible prediction of the data in the validation set.

```
# use model 7, this time on the entire edx dataset and the validation set
# Reset everything except main data sets
rm(list=setdiff(ls(), c("edx","validation")))
# add "week" column to both datasets to allow grouping and joining
validation <- validation %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week"))
edx <- edx %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week"))
# the following is identical to Model 7 above
lambdas <- seq(0, 10, 0.25)
rmses_final <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
```

```
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_t <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(week) %>%
    summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+l))
  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "week") %>%
    mutate(pred = mu + b_i + b_u + b_t) %>%
    .$pred
  return(RMSE(predicted_ratings, validation$rating))
})
# optional: display lambda optimisation graph and minimised lambda (5.5)
# lambdas[which.min(rmses_final)]
# qplot(lambdas, rmses_final)
# corresponding minimised RMSE is saved
final_model_rmse <- min(rmses_final)
```

```
# this displays the final RMSE value - 0.8646938
cat("Final Model Performance:", final_model_rmse)
```

```
## Final Model Performance: 0.8646938
```

The final RMSE value, when we attempt to predict ratings from the validation set using our most effective model, is **0.8646938**.

Conclusion

In summary, we determined that the most effective model for predicting the ratings in the validation set was the one which used individual movie and user means, took timestamp into account, and was regularised to compensate for variance in the amount of ratings per user/movie. Although the improvements to RMSE from adding time and performing the regularisation were minor, they still did provide improvements over the model which only used movie & user means, and the RMSE of the final model was satisfactory.

One limitation of this project was not using genre in the predictive model. Exploratory analysis revealed that genre did have an effect on ratings - with comedies being rated lower than other genres - but as it is a categorical factor, incorporating it into the model alongside the numeric predictors proved to be problematic.

Future research could take into account even more information from the data set. Possibly the most suitable candidate would be the user-defined "tags" given to each movie. As these descriptors are given to movies by the userbase, they may provide the best means of predicting ratings - the model could simply look for movies with similar tags to those which a user has already rated highly. Thus, each user could be expressed as a combination of preferred tags, weighted according to their preferences as revealed by their history of ratings. This would likely result in a highly effective predictive model - assuming that enough movies have been thoroughly tagged by the userbase to make it function properly.