Capstone Report Movielens

C. Heather

Introduction

This document is a final report for the HarvardX Data Science: Capstone (PH125.9x) course, utilising the movielens data set to construct a movie recommendation system based on user ratings.

After import and formatting, the movielens dataset contains movie ratings between 1 and 5 submitted by individual users. In addition to individual user ID (`userId`), movie ID (`movieId`) and rating (`rating`), the time stamp of review submission (`timestamp`), movie title (`title`) and genre tags (`genres`) were included in the data set.

The aim of this project is to allow accurate prediction of the user review for a specific movie, based on the movie reviews presented in the data set. The root mean square error (RMSE) of predictions, using a validation subset, is used as the measure of loss of the prediction algorithm.

The key steps taken in achieving this outcome were: *Data cleaning and formatting* Exploratory data analysis *Model development* Model fitting *Running the model against the `validation` data set

Methods and Analysis

Data preparation

The movielens data set was imported and cleaned using the code provided in the course material. This code further partitions the data into working (`edx`) and validation (`validation`) data sets. All work performed after this, other than final validation, was performed on the `edx` data set.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################


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



# Download, clean and combine data sets into movielens data frame

### Because issues were encountered with Markdown Knit-ing due to RStudio Server proxy settings,
### the data folder "ml-10M100K" was downloaded (unmodified) to a local folder for Markdown Knit-ing.

### Original download code was uncommented in submitted files to enable reproducibility .

# Download code provided
#
```

```
# 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")


# Modified code to read data from locally downloaded folder
ratings <- fread(text = gsub("::", "\t", readLines("data/ml-10M100K/ratings.dat")),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines("data/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))

# End modified code


movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens 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)

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

In addition to the columns provided, the release year of each movie is included in parentheses in the `title` character string. This was extracted into a new column `release_year`.

```
# Create new column with release year in edx

edx<- edx %>%
  mutate(release_year = str_extract(title, "(?<=\\()[:digit:]{4}(?=\\))"))
```

In the `edx` data set movie genres are supplied in the `genres` column. However most movies have multiple

genres assigned in one column, separated with a delimiter. In order to assess genre effect on rating, the genres were split at the "|" delimiter and a new row was created for each. This means a number of movie/ rating combinations are represented multiple times. From the resulting data set, a genre specific bias could be determined.

```
# create separate datafram with split genres into multiple rows, where necessary

edx_genres <- separate_rows(edx, genres, sep = "\\|") %>%
  select(rating, genres, userId, movieId)
```

Exploratory data analysis

The `edx` dataset contains 9000055 entries, representing individual user ratings for a particular movie.

```
# Number of ratings
nrow(edx)
```

```
## [1] 9000055
```

User effects

There dataset contains 69878 users, and ratings for 10677 movies.

```
# Number of users
print("Number of users")
```

```
## [1] "Number of users"
```

```
edx %>% group_by(userId) %>% summarise(n()) %>% nrow()
```

```
## [1] 69878
```

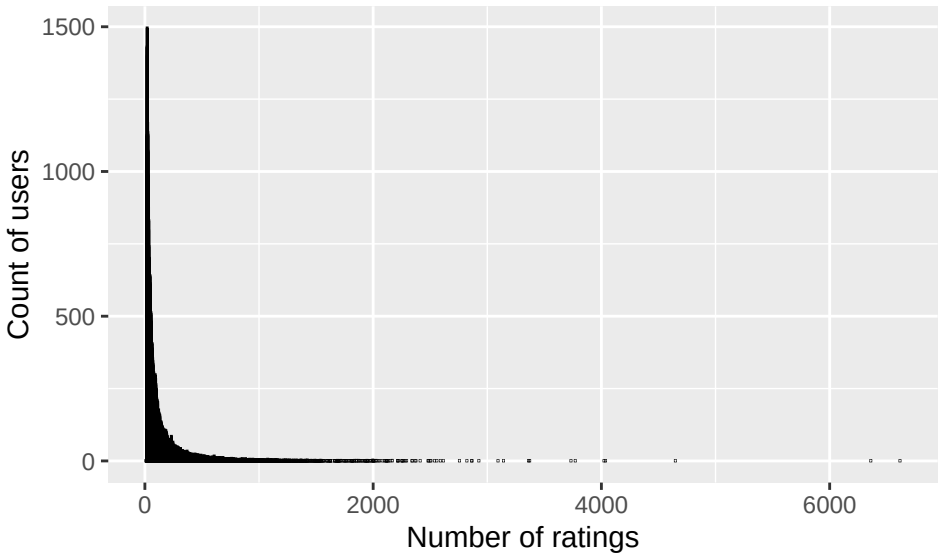
```
# Number of movies
print("Number of movies")
```

```
## [1] "Number of movies"
```

```
edx %>% group_by(movieId) %>% summarise(n()) %>% nrow()
```

```
## [1] 10677
```

The mean number of ratings per user is 128.7966885, with distribution as seen in the chart below. This demonstrates significanty outliers, with the most active user providing 6616 ratings!

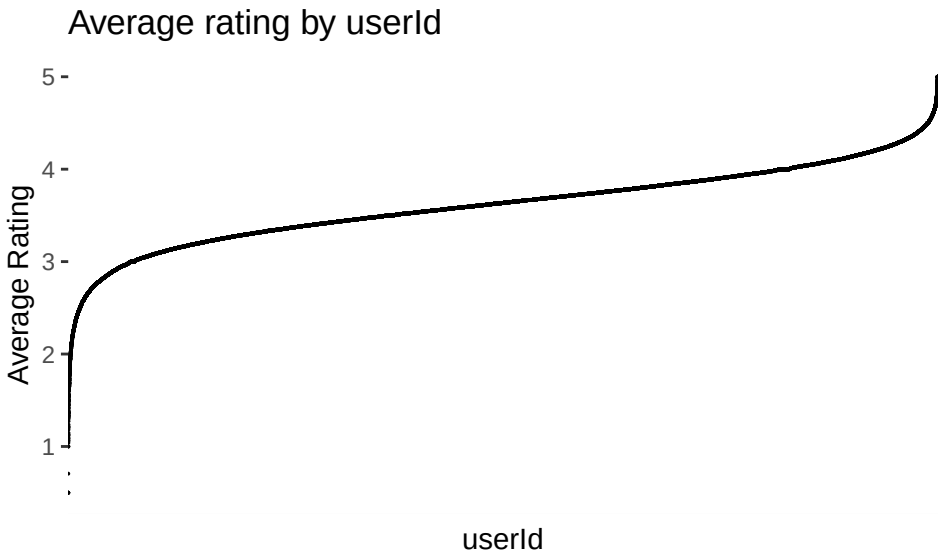


The overall average rating `mu` is 3.5124652.

```
# mean rating
mu <- mean(edx$rating)

mu
```

```
## [1] 3.512465
```

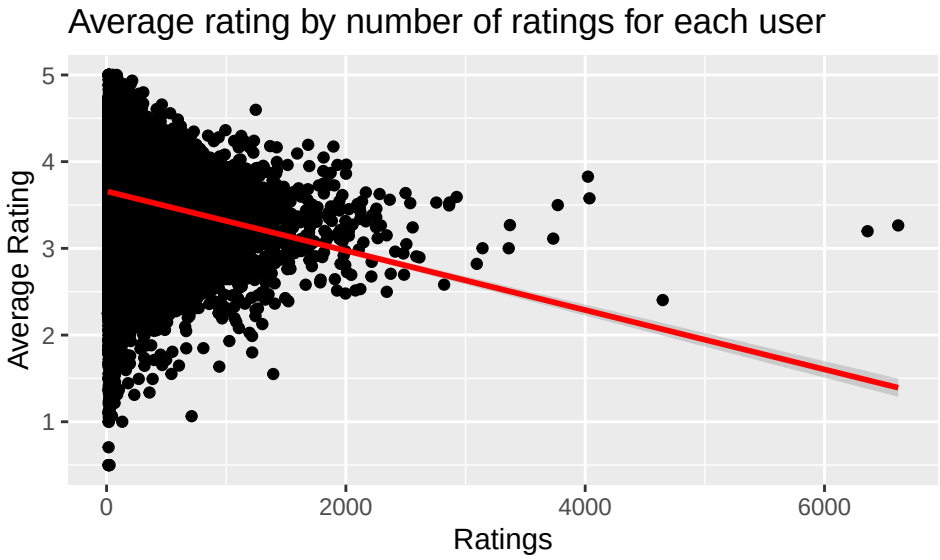
When examining average rating per user, it appears clear that some users tend to give higher ratings than others. This will be accounted for by normalising the user scores during model development.



When accounting for number of reviews per user, a trend towards lower scores with more reviews is apparent, suggesting that users with fewer ratings tend to give higher ratings to the movies they do rate, while users with more ratings tend to be lower.

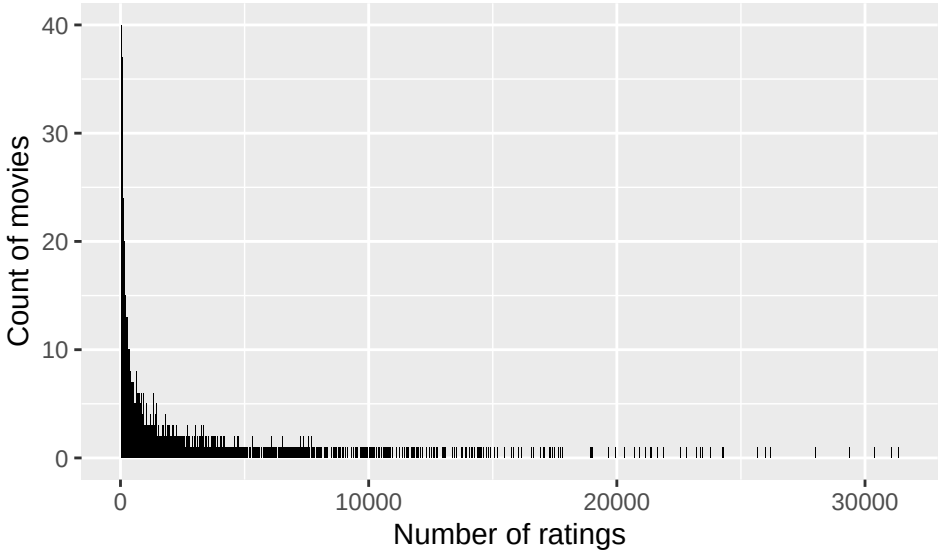
```
# Correlation of number of rating per user and rating chart
edx %>% group_by(userId) %>%
  summarise(avg_rating = mean(rating), num_ratings = n()) %>%
  ggplot(aes(num_ratings, avg_rating)) +
  geom_point() +
```

```
  geom_smooth(method = "lm", color = "red")+
  labs(
    title = "Average rating by number of ratings for each user",
    x = "Ratings",
    y = "Average Rating")
```

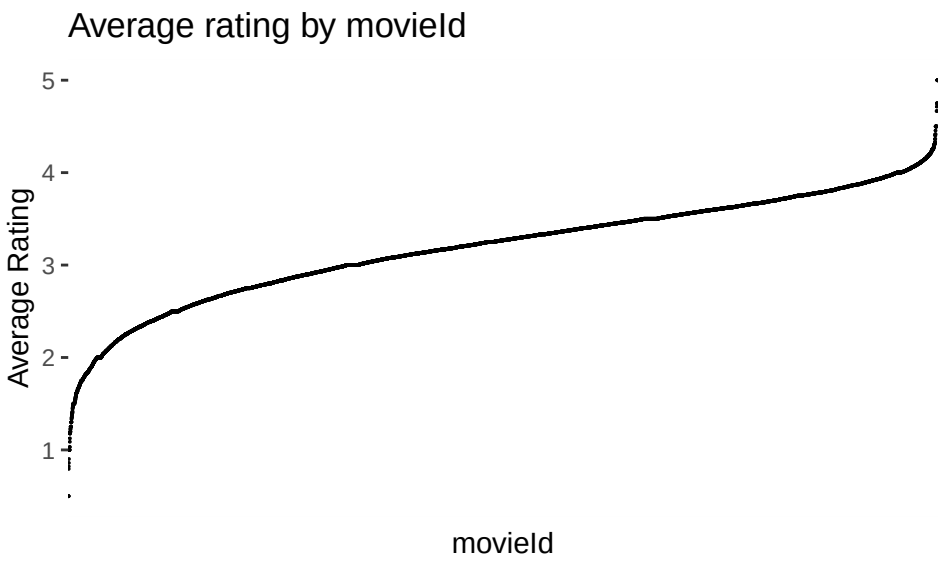


Movie effects

The average number of ratings per movie in `edx` is 842.9385595. This again shows that some movies appear to me more popular/ frequently rated than others, the most popular movie appears to be Patton (1970), with 31362 ratings.



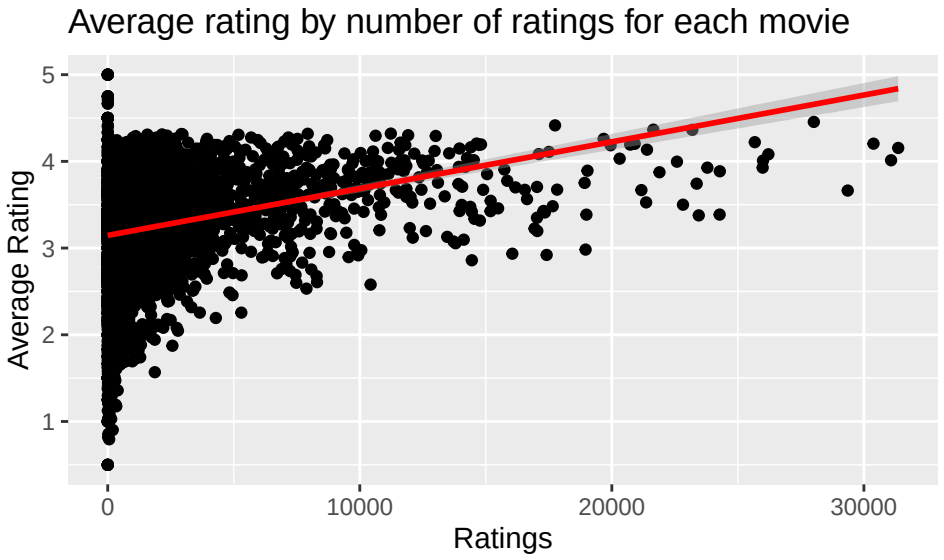
Examining the average rating for each movie unsurprisingly shows that some movies appear to be more popular than others.



This implies that movie effect is likely to be a significant parameter.

When correlating number of reviews for each movie with its mean rating, movies with more ratings tend be rated higher than those with fewer ratings.

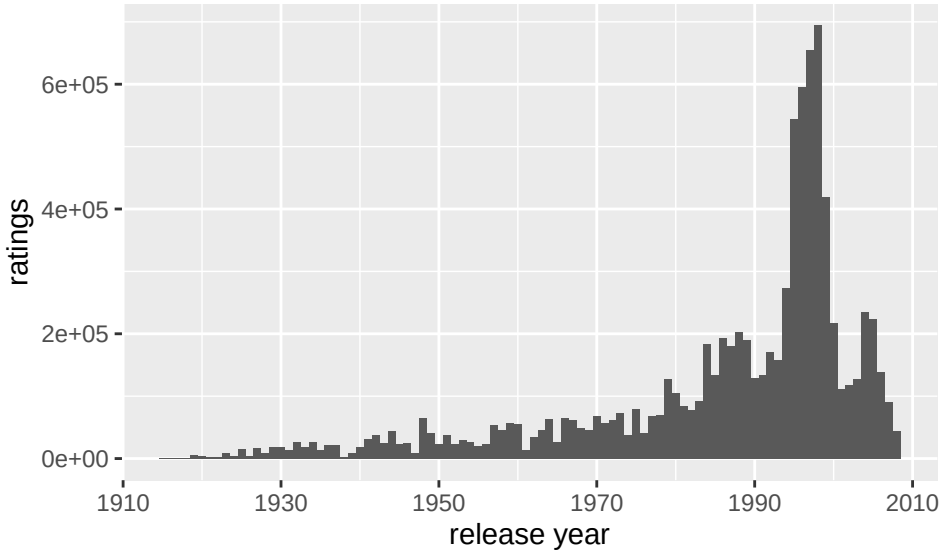
```
# Correlation of number of ratings per movie and rating chart
edx %>% group_by(movieId) %>%
  summarise(avg_rating = mean(rating), num_ratings = n()) %>%
ggplot(aes(num_ratings, avg_rating)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red")+
  labs(
    title = "Average rating by number of ratings for each movie",
    x = "Ratings",
    y = "Average Rating")
```



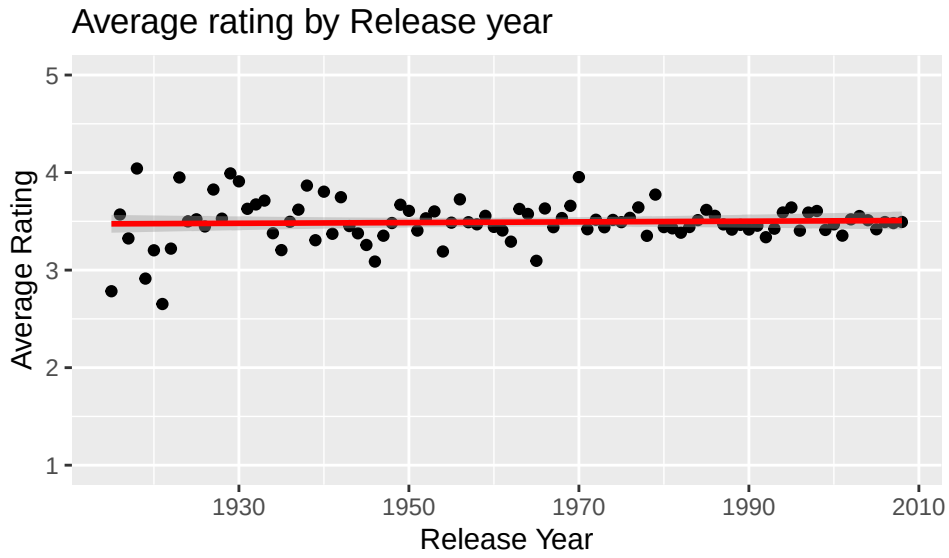
Release year

When examining release year, more recent movies appear to be more frequently reviewed.

```
# Number of reviews by release year chart
ggplot(edx, aes(as.numeric(release_year))) +
  geom_bar() +
  labs(x = "release year", y = "ratings")
```

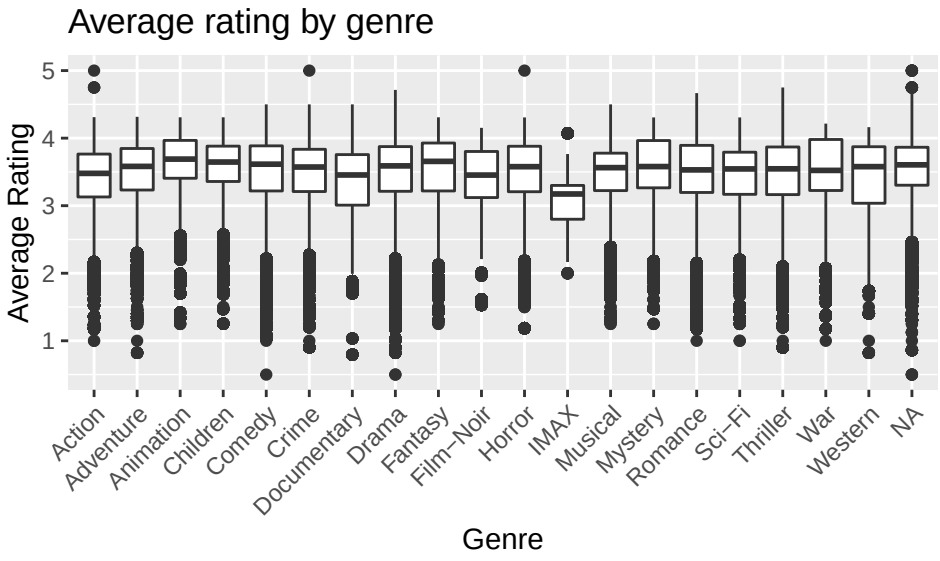


The release year also appears not to have a significant effect on average rating. The correlation coefficient of this effect is NA. Release year is therefore unlikely to have a substantial predictive effect.



Genre effects

When examining the average rating of each genre, there appears to some variation, with some genres receiving higher ratings than others, though this effect does not appear particularly pronounced (perhaps with the exception of IMAX). Genre in itself does not seem to have a significant effect on ratings.



Model development

The root mean squared error (RMSE) was used to assess accuracy of rating predictions obtained, using the following function

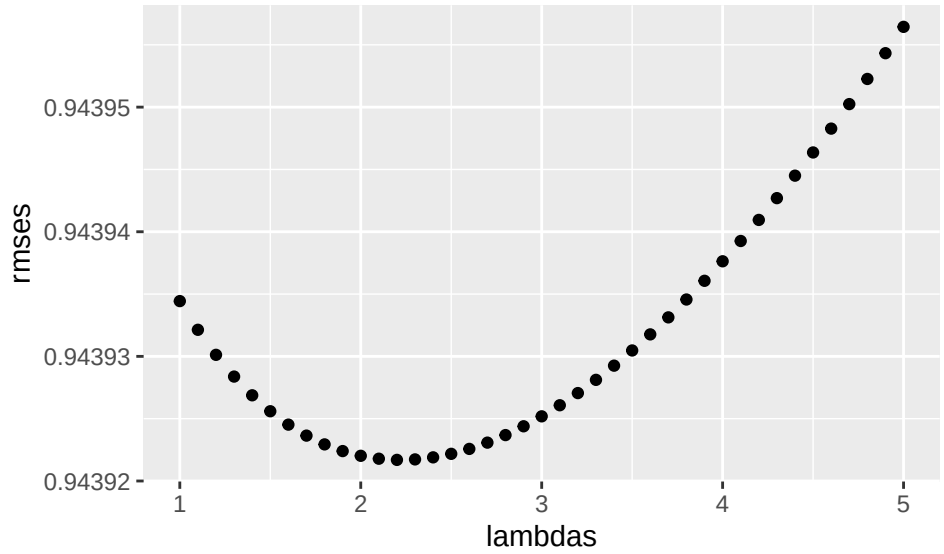
```
# RMSE function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Model proposed in course materials

The model proposed in the course materials accounts for user bias and movie bias, subtracting average user rating and average movie rating from `mu`. This allows predictions based on movie and user combinations. The model developed in the course materials is included here for reference.

```
#########################################################
## User and Movie bias model from course materials ##
##          Included for reference                 ##
#########################################################

#Partition edx into test (20%) and train (80%) sets
set.seed(755)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

#ensure userId and movieId from train_set are in test_set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")


# Select range for tuning of regularisation factor
lambdas <- seq(1, 5, 0.1)
```

```r
# Create tuning function returning rmses
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 - mu - b_i)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)


  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```




```r
# select optimum regularisation parameter

l <- lambdas[which.min(rmses)]

# best RMSE with this method

rmse_1 <- min(rmses)

rmse_1
```

```
## [1] 0.9439217
```

```
rm(test_index, test_set, train_set)
```

The RMSE for this method was 0.9439217.

This RMSE is not nearly close enough to satisfy the netflix challenge, so another approach may be necessary.

Matrix Factorisation

We would expect that due to differing tastes not everyone will enjoy the same movie equally. Some will enjoy comedies more than dramas, for example. The average rating of a movie is therefore likely to only partially inform the rating that an individual user will give it. The rating will also depend on whether the movie is in a genre that the user enjoys.

Collaborative filtering is one approach to address this issue. It relies on grouping users based on the similarity of their ratings. Once a user is placed with similar users, it is possible to predict their likely rating of a previously unrated movie based on the ratings of others in this "neighborhood".

When applying matrix factorisation dimension reduction is frequently used to reduce the computational requirements with large data sets. In this case dimension reduction results in individual movies being grouped into groups with similar characteristics (latent factors - which are unnamed by the algorithm but may correspond to "real life" observed features, such as "drama" or "romantic comedy"). The optimal number of final dimension can be determined during tuning.

The `recosystem` package provides tools to develop matrix factorisation based recommendations and rating predictions. The package allows import of data (testing and training sets), parameter tuning

Data formatting for recosystem The recosystem package accepts data from a variety of sources. In this case data from the `edx` and `validation` data frames are imported into `train_set` and `test_set` DataSource objects using the `recosystem::data_memory()` function.

```
### Data preparation

# create data objects using recosystem::data_memory() function

train_data <- data_memory(user_index = edx$userId,
                          item_index = edx$movieId,
                          rating = edx$rating)

test_data <- data_memory(user_index = validation$userId,
                         item_index = validation$movieId,
                         rating = validation$rating)

# Memory cleanup
rm(edx)
```

Tuning of model A number of tuning parameters are specified and optimal values determined by cross validation. This is achieved using the `$tune()` method.

```
### Setting tuning parameters and tuning model

# Specify tuning parameters
parameters <- list(dim = c(10, 20, 30),  # number of latent factors
                   lrate = c(0.1, 0.2), # learning rate - the step size in gradient descent
                   costp_l1 = 0, # Set to 0. L2 regularisation used.
                   costq_l1 = 0, # Set to 0. L2 regularisation used.
                   costp_l2 = c(0.01, 0.1), # L2 regularisation cost for user factors.
```

```
                    costq_l2 = c(0.01, 0.1), # L2 regularisation cost for movie factors.
                    niter = 10, # number of iterations
                    loss = "l2") # loss function. Specifying squared error ("l2")


# create empty recommender object
recommender <- Reco()

# Set training options using tuning parameters
opts = recommender$tune(train_data, opts = parameters)
```

The tuning parameters resulting in the smallest loss are stored in `opts$min`. These parameters will be passed to the `recommender$train()` method.

```
# Optimal tuning parameters
opts$min
```

```
## $dim
## [1] 30
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.7973644
```

Next, using the optimal tuning parameters the model will be trained on the `train_data` set using the `recommender$train()` method.

```
### Train model

# train the model using best options from tuning data "opts$min"
recommender$train(train_data, opts = c(opts$min, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9717   1.2007e+07
##    1       0.8714   9.8719e+06
##    2       0.8380   9.1608e+06
##    3       0.8162   8.7417e+06
##    4       0.8009   8.4630e+06
##    5       0.7893   8.2725e+06
##    6       0.7799   8.1206e+06
##    7       0.7720   8.0037e+06
##    8       0.7652   7.9069e+06
```

```
##    9       0.7594   7.8288e+06
##   10       0.7543   7.7631e+06
##   11       0.7497   7.7060e+06
##   12       0.7456   7.6548e+06
##   13       0.7418   7.6095e+06
##   14       0.7384   7.5731e+06
##   15       0.7351   7.5361e+06
##   16       0.7322   7.5065e+06
##   17       0.7295   7.4784e+06
##   18       0.7270   7.4534e+06
##   19       0.7247   7.4298e+06
```

The `recommender$predict()` method is then used to create a vector of predictions from the `test_data` (original `validation` data set), and our RMSE function is run using this vector, storing the result in `rmse_2`.

```
### Determine RMSE using  predictions from recommender$predict method and validation$rating vector

# Create vector of predictions for each user
pred_rvec <- recommender$predict(test_data, out_memory())

# Run RMSE function
rmse_2 <- RMSE(validation$rating, pred_rvec)

rmse_2
```

```
## [1] 0.7826351
```

Results

The final RMSE using matrix factorisation with the `recosystem` package produced was 0.7826351. This was a substantial improvement on the RMSE obtained from the user/ movie bias model - 0.9439217. The training parameters used were 30 latent factors, user regularisation cost of and movie regularisation cost of 0.1.

The result demonstrates that preference based clustering of users is a powerful tool in predicting individual ratings. For large data sets such as these, the emergence of latent factors during dimension reduction in particular is highly useful in implementing predictive algorithms with limited computing capacity.

Conclusions

The movielens dataset examined contained 10 million reviews. A tuned matrix factorisation algorithm was used to predict user ratings based on ratings of similar users. This approach resulted in a RMSE loss of 0.7826351.

The data set provided contained relatively few features that were correlated with ratings, limiting the utility of other clustering approaches, such as random forests, while being computationally intensive. A different approach to the one presented here could have been item/ movie based collaborative filtering, based on identifying movies which are similar, rather than users. This was not explored on this occasion.

Further improvement in RMSE might have been possible with further tuning of parameters. The number of dimensions/ latent factors after tuning was at the upper limit of the values provided to the tuning function, so a larger number of dimensions may have resulted in better outcomes (at the risk of over-fitting).

In view of this, while the RMSE was well within the parameters required for this project, further optimisation of the methods used, could result in even better outcomes, and might be considered for future exploration outside of this project.