MovieLens project: A movie recommendation system

Andreas Maos

31 August, 2019

Contents

1 Introduction

1.1 Movie recommendation systems

Movie recommendation systems have been around for a while and are being used by numerous large streamig services and companies such as Netflix and Amazon Prime Video. These systems use several different methods to make personalized recommendations about movies or TV shows to customers. Their goal is for these recommendations to be as appealing as possible to each individual customer. Some of the methods used include **User-Based Collaborative Filtering (UBCF)** and **Item-Based Collaborative Filtering (IBCF)**.

UBCF identifies users that thave the same taste in movies. For example, both users A and B have seen movie X and have given it the same rating, indicating that they have similar tastes. User A also saw movie Y, liked it and gave it a high rating. User B has not seen movie Y but since he has a similar taste to user A, the system will recommend movie Y to user B. The underlying machine learning algorithm of UBCF is the K-nearest neighbors (KNN) algorithm. KNN identifies the K most similar users to the *active user* (in the above example user B) and recommends to him movies that those K similar users liked.

IBCF on the other hand, identifies items, in this case movies, that are similar to each other. For example, if user A liked movie X then an IBCF system will recommend to user A movies that are similar to movie X. Consider movies X, Y and Z that belong to the genres Comedy, Animation/Comedy and Drama respectively.

When user A rates movie X highly, the system will recommend movie Y which is similar to the movie that he liked, instead of movie Z that belongs to a completely different genre.

1.2 The project and dataset

This project constitues one of the final parts of the Professional Certificate in Data Science by HarvardX. The aim of this project was to develop a movie recommendation system. To achieve this, the MovieLens dataset created by GroupLens Research was used. Several MovieLens datasets were available but for this project the MovieLens 10M Dataset was used. This contains about 10 million movie ratings and was released in 2009.

To obtain this dataset in an appropriate format to begin the project, a piece of code provided by EdX was run (see Appendix section 5.1). After running this code two data frames were obtained, the **edx set** which was used to develop the algorithm, and a **validation set** which was used to test the algorithm.

To evaluate how close the predictions were to the true values in the validation set, the **Root Mean Square Error (RMSE)** was calculated.

The **edx set** had 9000055 rows, 6 columns, 69878 users and 10677 movies. The **validation set** had 999999 rows, 6 columns, 68534 users and 9809 movies.

The two sets had the same structure. In both sets, observations were found in the rows. These correspond to individual movie ratings. Features were found in the columns and these included userId, movieId, rating, timestamp, title, genres. Each user and movie were assigned their own unique ID number, *userId* and *movieId*. The *rating* column contained the scores, values between 0 and 5 in 0.5 increments, assigned by each user to each movie. The *timestamp* column contained information about the date and time each rating was submitted. The information was represented as the **UNIX Epoch Time** which is the number of seconds that had elapsed since 00:00:00 Thursday, 1 January 1970. Part of the analysis included converting this in a more understandable format. The *title* column contained the title of the movie and the year it was released in parenthesis (i.e. My movie title (1995)). Finally, the *genres* column provided information about the genres in which a movie belonged to.

These features provided useful information about each observation and played a pivotal role in developing the movie recommendation algorithm. In the analysis that follows 3 methods will be explored, regression, UBCF and IBCF.

2 Analysis

2.1 RMSE function

Calculating the RMSE was an integral part of the algorithm development process as this was the measure used to evaluate the algorithm's performance. Since this computation was done multiple times throughout the project, it was worth defining a function that would easily compute it.

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

The above code performs the following calculation:

$$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

with $y_{u,i}$ corresponding the actual rating of user u for movie i and $\hat{y}_{u,i}$ corresponding to the model's prediction.

2.2 Data wrangling

The edx and validation sets were in tidy format but required some more wrangling. Firstly, the year a movie was released was extracted from the *title* column. The regex pattern `\\s\\(\\d{4}\\)` along with the functions `str_extract()` and `str_replace()` were used to detect and extract the year and place it in a new column named *release_year*. Then, a new column called *rating_period* was created. This contained the difference between the year the dataset was released, 2009, and the year a movie was released, found in the *release_year* column. Finally, the *date* column was converted from the UNIX Epoch Time format to a more readable format. This was achieved using the `as_datetime()` function from the **lubridate** package. These dates were accurate to the second and hence there were many different ones. To reduce the number of distinct dates, they were rounded to the nearest week using lubridate's `round_date()`function.

2.3 Data exploration and visualization

From this point onwrads, **only the edx set was used** to build and test the model and the validation set was only used at the end to evaluate the final model.

Several plots were created to explore basic data characteristics that would help in building the recommendation model.

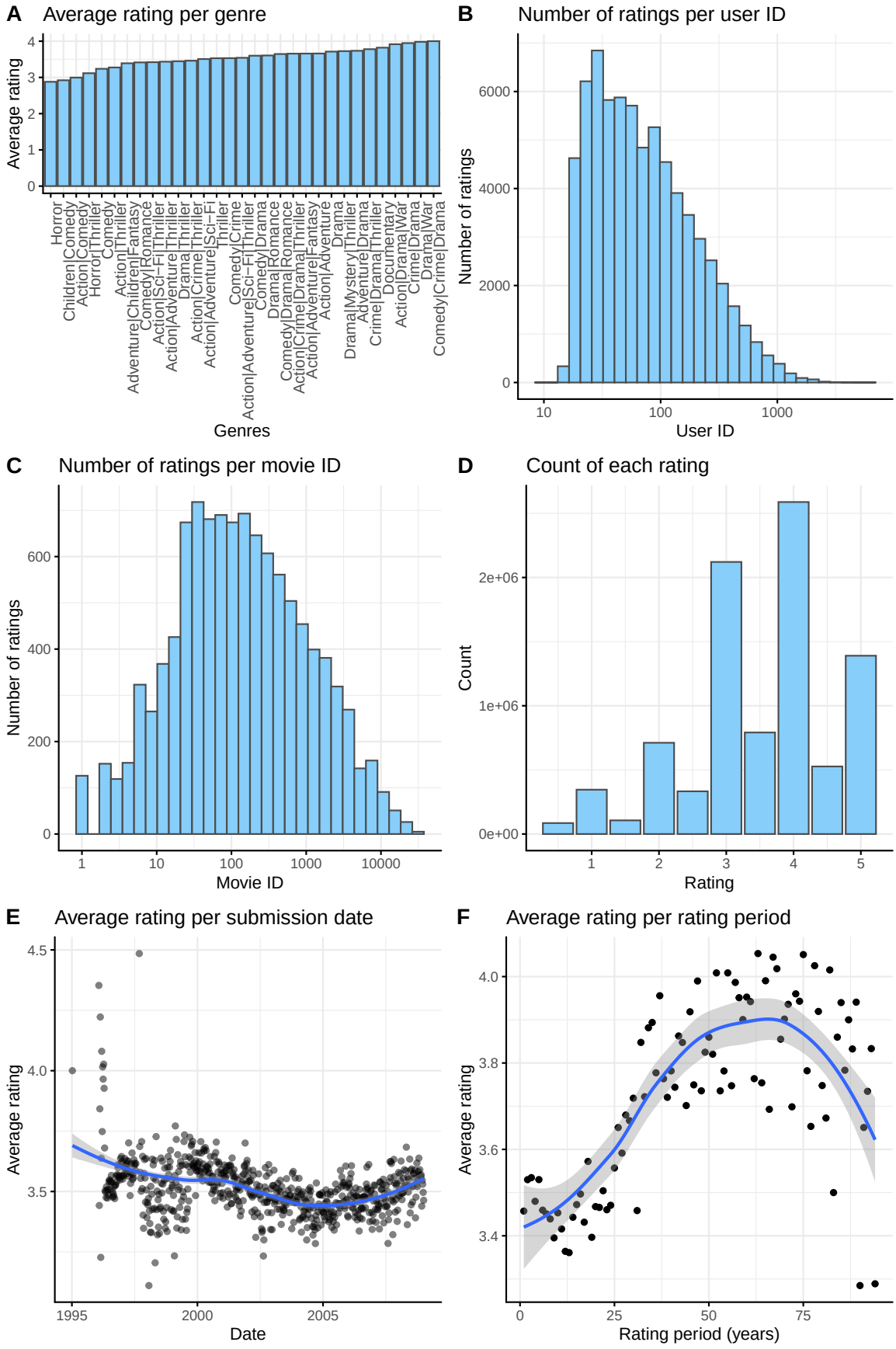
Plot **A** shows the average rating for some of the most popular genres. Genres with 50000 movies or more were kept and the average rating for each genre was calculated.

Plot **B** shows the number of ratings per user. Plot **C** shows the number of ratings per movie. In both cases there are users or movies with very few ratings. These observations could introduce uncerainty and negatively affect the model resulting in unreliable predictions.

Plot **D** shows the frequecy of each possible rating (0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5). One can see that whole number ratings are more common than ratings with half points. It is also evident that ratings of 3 and 4 are the most common.

Plot **E** shows the average rating per rating submission date (rounded to the nearest week). This shows that the date a rating was submitted had a minor effect on the rating. Therefore, it might not be advantageous for this feature to be included in the model. Further exploration of this effect will take place downstream.

Plot **F** shows the average rating per rating period. Rating period is the number of years between the year a movie was released and 2009, the year the MovieLens 10M Dataset was released. A larger number for the rating period corresponds to movies that were released in earlier years. It seems that many movies that were released 37 or more years before 2009 received a higher rating. A reason for this could be that movies that have been around for longer had more time, and therefore more chances, to be seen by more people or to gain popularity and hence to receive a lot of high ratings. This could suggest that the rating period could provide useful information when making predictions.


A
Average rating per genre
Average rating
Genres
Horror
Children|Comedy
Action|Comedy
Horror|Thriller
Comedy
Action|Thriller
Adventure|Children|Fantasy
Comedy|Romance
Action|Sci-Fi|Thriller
Action|Adventure|Thriller
Drama|Thriller
Action|Crime|Thriller
Action|Adventure|Sci-Fi
Thriller
Comedy|Crime
Action|Adventure|Sci-Fi|Thriller
Comedy|Drama
Drama|Romance
Comedy|Drama|Romance
Action|Crime|Drama|Thriller
Action|Adventure|Fantasy
Action|Adventure
Drama
Drama|Mystery|Thriller
Adventure|Drama
Crime|Drama|Thriller
Documentary
Action|Drama|War
Crime|Drama
Drama|War
Comedy|Crime|Drama
B
Number of ratings per user ID
Number of ratings
User ID
C
Number of ratings per movie ID
Number of ratings
Movie ID
D
Count of each rating
Count
Rating
E
Average rating per submission date
Average rating
Date
F
Average rating per rating period
Average rating
Rating period (years)

2.4 Building the regression model

With the MovieLens 10M Dataset containing about 10 million ratings, it was quite hard to fit complicated machine learning models. Even performing simple calculations or data wrangling operations would take several seconds to minutes on a standard laptop, while consuming a lot of CPU capacity and RAM. Hence, to mitigate part of this effect, the edx set was further split into a training set and a test set containing 75% and 25% of the observations respectively. This was done using the `createDataPartition()` function from the **caret** package. The training set was used to build and fit the model and the test set was used to test the model's accuracy using the RMSE function defined above. Only when the model was optimized as much as possible, the validation set was used to evaluate the model.

2.4.1 Just the average

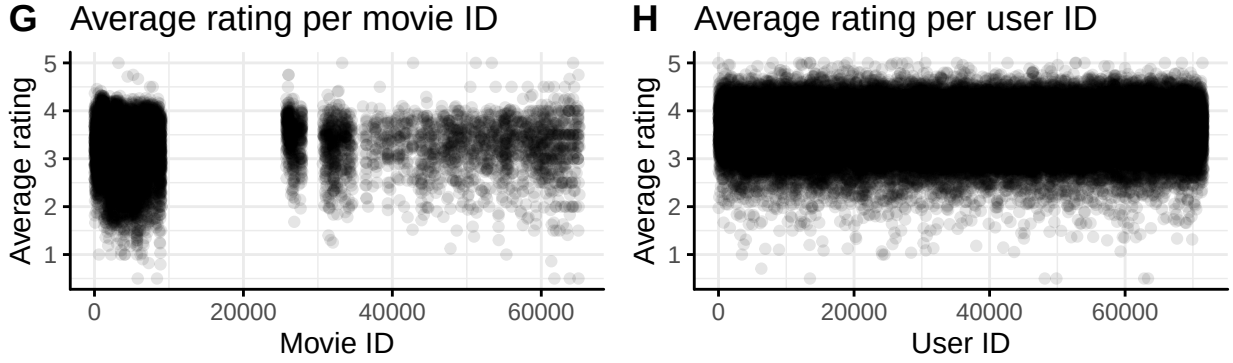
The simplest idea, that was used as the starting material for the model, was that all movies were predicted to have a rating equal to the average movie rating in the training set with any differences explained by random variation ($\epsilon_{u,i}$).

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

The average rating was 3.5124 while the RMSE for this model was **1.06**. This was not very good but it was a solid starting point.

2.4.2 Movie and user model

It is a fact that different movies are rated differently and therefore some movies will generally be rated higher than others. This is called the **movie-specific effect**. Also, users differ in how they rate movies and hence there is substantial variability across users as well. For example, some users will consistenly give low or high ratings while others are somewhere in the middle. This is called the **user-specific effect**. These effects can be seen from the plots below.



These considerations allow the addition of a term b_i to the model. This represents the average rating for movie i. They also suggest that a term b_u can be added, representing the user-specific effect (the average rating for each user). The model now looks like this:

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

To calculate b_i the average of $Y_{u,i} - \mu$ was used.

To calculate b_u the average of $Y_{u,i} - \mu - b_i$ was used.

The RMSE for this model was **0.8662**. There is a clear improvement but the model can be further optimized.

2.4.3 Sanity check

Title	b_i	Number of ratings
Hellhounds on My Trail	1.487643	1
If You Only Knew	1.487643	1
Intended, The	1.487643	1
Satan's Tango (Sátántangó)	1.487643	1
Shadows of Forgotten Ancestors	1.487643	1
Fighting Elegy (Kenka erejii)	1.487643	1
Sun Alley (Sonnenallee)	1.487643	1
Maradona by Kusturica	1.487643	1
Class, The (Entre les Murs)	1.487643	1
Blue Light, The (Das Blaue Licht)	1.487643	1
30 Years to Life	-3.012357	1
Besotted	-3.012357	2
Camera Obscura	-3.012357	1
Hi-Line, The	-3.012357	1
Grief	-3.012357	1
Accused (Anklaget)	-3.012357	1
Confessions of a Superhero	-3.012357	1
War of the Worlds 2: The Next Wave	-3.012357	2
Disaster Movie	-2.783191	24
SuperBabies: Baby Geniuses 2	-2.728266	44

The table above shows the top (green) and worst (red) 10 movies by b_i along with the number of ratings for each one. It is clear that the number of ratings is very small. This introduces a lot of uncertainty resulting in smaller or larger estimates of b_i. These noisy estimates result in untrustworthy predictions. Also, looking at the movie titles one can sense that something is wrong as it can be expected to see more popular movies in the green rows.

2.4.4 Regularized movie and user model

Regularization can be used to overcome the variability introduced by estimates coming from small samples by penilizing large estimates that come from such samples. The equation the model minimizes now becomes:

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

with the values of b that minimize the above equation being calculated using:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu}),$$

with n_i being the number of ratings for movie i. The larger the value of λ, the more the estimate shrinks.

λ is a tunable parameter and hence a range of different values needed to be tested in order to decide which one minimized the RMSE. In this model, values from 4 to 6 in 0.2 increments were tested in an `sapply()` function to generate a vector of RMSEs and from that choose the best λ.

The optimum λ for this model was **4.8** and the RMSE was **0.8655**. Additional features be considered to further improve the model!

2.4.5 Validate the regularization results

The same sanity check as before was performed on the regularized results. The movies shown in the table now make much more sense to have a high or low b_i value and the number of ratings is significantly higher than those of the movies previously listed. This proves that regularization improved the estimates.

Title	b_i	Number of ratings
Shawshank Redemption, The	0.9417453	20982
Godfather, The	0.9015225	13334
Usual Suspects, The	0.8511899	16248
Schindler's List	0.8491857	17450
Casablanca	0.8135839	8442
Rear Window	0.8120905	5972
Sunset Blvd. (a.k.a. Sunset Boulevard)	0.8103829	2208
Paths of Glory	0.8091700	1176
Third Man, The	0.8083790	2200
Double Indemnity	0.8070705	1597
From Justin to Kelly	-2.5119765	151
SuperBabies: Baby Geniuses 2	-2.4599123	44
Pokémon Heroes	-2.3700197	101
Disaster Movie	-2.3193255	24
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie)	-2.2978690	147
Glitter	-2.2778087	247
Carnosaur 3: Primal Species	-2.2648341	52
Gigli	-2.2586660	228
Barney's Great Adventure	-2.2242318	162
Turbo: A Power Rangers Movie	-2.1393446	305

2.4.6 Regularized Movie, User, Genres, Rating Period and Date Effect Model

Three new features were introduced into the model. The genres a movie belongs to, b_g, the number of years between 2009 (when the MovieLens 10M Dataset was released) and the year a movie was released, b_r and the date the rating was submitted rounded to the nearest week, b_t. These could be found in the *genres*, *rating_period* and *date* columns respectively. The model now looks like this:

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

The optimum λ for this model was **5** and the RMSE was **0.8648**.

To validate that the value of λ obtained above was the best one, **cross-validation** was used. Four-fold cross-validation was used to split the data into 4 folds, each consisting of 75% training data and 25% test data. Essentially, the training set obtained by splitting the edx set at the beginning, was further split into 4 folds with a training and test set in each fold. For each fold, a range of lambdas was used to make predictions using the model discussed above. The lambda that minimized the RMSE was returned for each fold and at the end the mean of all the lambdas returned was computed. This verified that the value of λ obtained above was the optimal one (see Appendix section 5.2 for the cross-validation code).

2.4.7 Model evaluation using the validation set

After gradually building and optimizing the model, it was fitted and evaluated on the validation set. One last adjustment took place before this. Originally, to make computations faster, the edx set was split into

training and test sets. While building the model, a value μ was used that represented the average rating for all observations in the **training set obtained from the edx set**. Therefore, before fitting and testing the final model on the validation set, **the average rating for all observations in the edx set** was used as μ. The value of λ remained the same as the one obtained above. Using the new value of μ, the RMSE on the validation set was **0.8641**.

2.5 The recommenderlab package

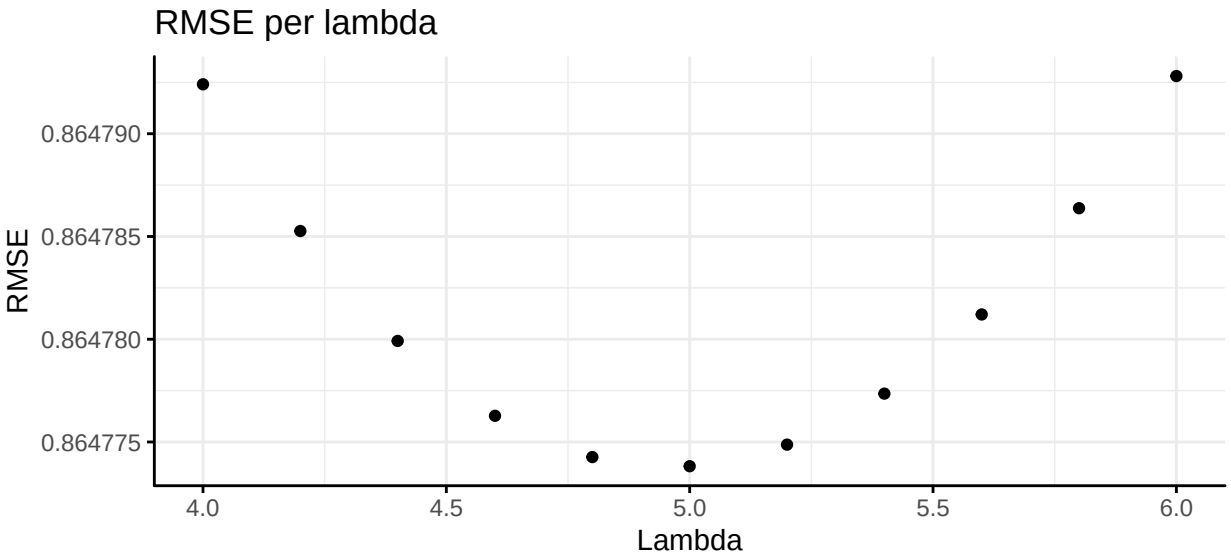
The `recommenderlab` package provides tools and methods to develop and test different recommender algorithms some of which include, User-based collborative filtering (UBCF), Item-based collborative filtering (IBCF), SVD with column-mean imputation (SVD) and Popular items (POPULAR). UCBF and ICBF were discussed at the beginning and were both explored in the following sections. Due to limited copmuting resources, the algorithms were applied on a subset of the edx set and were not used to make predictions on the validation set. Hence, the RMSEs obtained were not reported in the results. The code for running UCBF and ICBF using the recommenderlab package can be found in the Appendix.

Initially, the edx set was converted to a sparse matrix with user IDs in the rows, movie IDs in the columns and the matrix cells filled with the ratings. This means that many cells contained an NA as not all users had rated all movies. For both algorithms, 4-fold cross-valiadtion was used meaning that the matrix was split into 4 folds with each fold consisting of a 75% of the fold as the training set and 25% as the test set. A range of different values was used as the number of nearest neighbours and the RMSE for each one was calculated.

For UBCF, the number of nearest neighbours used was 80 and the RMSE was 0.8509 while for IBCF the number of nearest neighbours used was 700 and the RMSE was 0.9445. It is clear that UBCF returns a lower RMSE and is hence superior than IBCF in making predictions.

3 Results

The plot below shows how the RMSE for the final model changed with each λ. A concave up shape can be observed with a minimum point at $\lambda = 5$.



The table below shows how the RMSE changed with each of the model's evolution. For the models that utilised regularization, the optimum value of λ is also shown. **The first 4 rows of the table (white) show the RMSEs when fitting the model on the training set and evaulating it on the test set.**

As a reminder, the training and test sets were created by splitting the edx set into two subsets, 75% and 25% respectively.

The final row (green), shows the RMSE and λ for the fourth and best model fitted on the whole edx set and then **validated on the validation set**. Hence, **the minimum RMSE obtained using the final model was 0.8641 using a λ of 5**. The percentage improvement in the RMSE on the validation set was not calculated as all the previous percentages refer to the RMSE improvements on the training set, not the validation set.

Model	RMSE	Improvement (%)	Lambda
Just the average	1.0600	-	-
Movie and User	0.8662	18.2817	-
Regularized Movie and User	0.8655	0.0858	4.8
Regularized Movie, User, Genres, Rating Period and Date Effect Model	0.8648	0.0835	5
Validation set	**0.8641**	**-**	**5**

4 Conclusion

The aim of this project was to use different data science techniques to develop a movie recommendation system using the MovieLens 10M Dataset. To achieve this, a linear regression model was built. To optimize and evaluate the model, the edx set was split into training and test sets. Cross-validation was used to optimize the tunable parameter λ needed for regularization. After finalizing the model, it was used to make predictions on the validation set and the RMSE for those predictions was calculated to be **0.8641**. This value showed that a linear regression model with regularized effects on movies, users, genres, rating period and rating submission date (rounded to the nearest week) can quite accurately predict movie ratings and hence be used to predict if a user will like (high rating) or not (low rating) a given movie. Therefore, further steps could include using this model to predict user ratings on different movies and recommend movies predicted to receive a rating of more than 3.5, for example. Also, the UBCF and IBCF algorithms of the recommenderlab package were tested on a subset of the edx set. Reasonably good predictions were obtained with the RMSE for the UBCF algorithm being lower than that of the linear regression model. The UBCF and IBCF models could not be used to make predictions on the validation set due to memory constraints.

A significant challenge of this study was the memory limit. As the MovieLens 10M Dataset contained about 10 million observations, it was impossible to fit more sophisticated machine learning models, such as KNN, on the whole edx set. A way to fit such models could be to filter out some observations from the edx and validation sets, but this would come at the cost of losing information contained in the filtered out observations. Additionally, Principal Component Analysis (PCA) could be performed to identify hidden structure in the data not captured by the linear regression model. One could identify how many principal components were needed to capture most of the variation in the data, 90% for example, to reduce the dimensionality and then fit an appropriate machine learning model.

Overall, this project was very interesting and exciting. It provided a great opportunity to learn how recommendation systems work and how they are being used by major companies to benefit their business and serve customers. Several data science techniques had to be used to successfully complete the project and it was a good opportunity to hone existing skills and develop new ones. Looking forward to more data science projects like this one!

5 Appendix

5.1 EdX code to get the edx and validation sets

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

# 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")

movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[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)

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)
```

5.2 Choosing lambda using cross-validation

```
set.seed(1)

folds <- createFolds(y = train_set$rating, k = 4, list = TRUE, returnTrain = TRUE)

# THE CODE BELOW RUNS FOR SEVERAL MINUTES

preds <- lapply(X = names(folds), FUN = function(f) {

  lambdas <- seq(0, 7, 0.2)

  index <- folds[[f]]

  train_tmp <- train_set[index, ]

  temp <- train_set[-index, ]

  # Make sure userId and movieId in the test_tmp are also in train_tmp

  test_tmp <- temp %>%
    semi_join(train_tmp, by = "movieId") %>%
    semi_join(train_tmp, by = "userId")

  # Add rows removed from test_set back into train_tmp

  removed <- anti_join(temp, test_tmp) # generates a harmless message "Joining, by = c(..."

  train_tmp <- rbind(train_tmp, removed)

  rm(temp, removed)

  # Calculate the RMSE for each lambda

  rmses <- sapply(X = lambdas, FUN = function(l) {

    print(paste0("Lambda = ", l))

    b_i <- train_tmp %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu_rating) / (n() + l), n_i = n())

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

    b_g <- train_tmp %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - mu_rating - b_i - b_u) / (n() + l))
```

```
    b_r <- train_tmp %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_g, by = "genres") %>%
      group_by(rating_period) %>%
      summarize(b_r = sum(rating - mu_rating - b_i - b_u - b_g) / (n() + l))

    b_t <- train_tmp %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_g, by = "genres") %>%
      left_join(b_r, by = "rating_period") %>%
      group_by(date) %>%
      summarize(b_t = sum(rating - mu_rating - b_i - b_u - b_g - b_r) / (n() + l))

    predicted_ratings <- test_tmp %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      left_join(b_g, by = "genres") %>%
      left_join(b_r, by = "rating_period") %>%
      left_join(b_t, by = "date") %>%
      mutate(pred = mu_rating + b_i + b_u + b_g + b_r + b_t) %>%
      .$pred

    if (any(is.na(predicted_ratings))) {

      warning("ERROR! Ensure that the userId and movieId in the test_tmp are also in train_tmp!")

    } else {

      # Return a vector with the RMSEs

      return(RMSE(predicted_ratings, test_tmp$rating))

    } # else

  }) # nested sapply

  # Return a data frame with the lambda that minimizes the RMSE

  return(data.frame(lambda = lambdas[which.min(rmses)], rmse = min(rmses), stringsAsFactors = FALSE))

}) # lapply

names(preds) <- names(folds) # name the entries in the list

all_folds <- do.call(rbind, preds) # join the list entries into a single data frame

# Calculate the mean of all the different lambdas obtained and use that for predictions

lambda <- mean(all_folds$lambda)

lambda
```

5.3 UBCF

```
invisible(gc()) # clear unused memory

# Create a sparse matrix of the edx set.
# Each user gets a row.
# Each movie gets a column.

edx_fac <- edx %>%
  mutate(userId = factor(userId),
         movieId = factor(movieId))

sparse_ratings <- sparseMatrix(
  i = as.numeric(edx_fac$userId),
  j = as.numeric(edx_fac$movieId),
  x = edx_fac$rating,
  dims = c(length(unique(edx_fac$userId)),
           length(unique(edx_fac$movieId))))

rownames(sparse_ratings) = levels(edx_fac$userId)

colnames(sparse_ratings) = levels(edx_fac$movieId)

rm(edx_fac)

# Convert sparse_ratings to a recommenderlab sparse matrix

rating_matrix <- new("realRatingMatrix", data = sparse_ratings)

# Select users that have seen/rated many movies
# Select movies that have been watched/rated by many users

min_movies <- quantile(rowCounts(rating_matrix), 0.9)

min_users <- quantile(colCounts(rating_matrix), 0.9)

movie_ratings <- rating_matrix[rowCounts(rating_matrix) > min_movies,
                               colCounts(rating_matrix) > min_users]

set.seed(1)

# Use 4-fold cross-validation.
# 25 ratings of 25% of users are excluded for testing.
# the evaluationScheme() function runs for a while.

e <- evaluationScheme(movie_ratings, method = "cross-validation", k = 4, given = -25)

# Use algorithms from the recommenderlab package

# UBCF

nns <- seq(50, 80, 5)
```

```r
# Loop takes several minutes to run

rmses_ucbf <- sapply(X = nns, FUN = function(n) {

  print(paste0("UBCF: nn = ", n)) # to keep track of progress

  model_ubcf <- Recommender(getData(e, "train"), "UBCF",
                            param = list(normalize = "center", method = "Cosine", nn = n))

  # Make predictions

  preds_ubcf <- predict(model_ubcf, getData(e, "known"), type = "ratings")

  # Calculate the RMSE for the UBCF model

  rmse_ubcf <- calcPredictionAccuracy(preds_ubcf, getData(e, "unknown"))[1]

  return(rmse_ubcf)

}) # sapply

qplot(x = nns, y = rmses_ucbf)

nns[which.min(rmses_ucbf)] # best value of nearest neighbours to use

min(rmses_ucbf) # minimum RMSE
```

5.4 IBCF

```
set.seed(1)

ks <- seq(350, 700, 25)

# Loop takes several minutes to run

rmses_ibcf <- sapply(X = ks, FUN = function(k) {

  print(paste0("IBCF: k = ", k)) # to keep track of progress

  model_ibcf <- Recommender(getData(e, "train"), method = "IBCF",
                            param = list(normalize = "center", method = "Cosine", k = k))

  prediction <- predict(model_ibcf, getData(e, "known"), type = "ratings")

  rmse_ibcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]

  return(rmse_ibcf)

}) # sapply

qplot(x = ks, y = rmses_ibcf)

ks[which.min(rmses_ibcf)] # best value of k to use

min(rmses_ibcf) # minimum RMSE
```

5.5 Operating system

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          5.3
## year           2019
## month          03
## day            11
## svn rev        76217
## language       R
## version.string R version 3.5.3 (2019-03-11)
## nickname       Great Truth
```

5.6 R session information

```
## R version 3.5.3 (2019-03-11)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] data.table_1.12.2    caret_6.0-84         lattice_0.20-38
##  [4] forcats_0.4.0        stringr_1.4.0        dplyr_0.8.3
##  [7] purrr_0.3.2          readr_1.3.1          tidyr_0.8.3
## [10] tibble_2.1.3         ggplot2_3.2.1        tidyverse_1.2.1
## [13] kableExtra_1.1.0     knitr_1.24           RColorBrewer_1.1-2
## [16] cowplot_1.0.0        recommenderlab_0.2-5 registry_0.5-1
## [19] proxy_0.4-23         arules_1.6-4         Matrix_1.2-17
## [22] lubridate_1.7.4
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-141       webshot_0.5.1      httr_1.4.1
##  [4] tools_3.5.3        backports_1.1.4    R6_2.4.0
##  [7] irlba_2.3.3        rpart_4.1-15       lazyeval_0.2.2
## [10] colorspace_1.4-1   nnet_7.3-12        withr_2.1.2
## [13] tidyselect_0.2.5   compiler_3.5.3     cli_1.1.0
## [16] rvest_0.3.4        xml2_1.2.2         labeling_0.3
## [19] scales_1.0.0       digest_0.6.20      rmarkdown_1.15
## [22] pkgconfig_2.0.2    htmltools_0.3.6    rlang_0.4.0
## [25] readxl_1.3.1       rstudioapi_0.10    recosystem_0.4.2
## [28] generics_0.0.2     jsonlite_1.6       ModelMetrics_1.2.2
## [31] magrittr_1.5       Rcpp_1.0.2         munsell_0.5.0
## [34] stringi_1.4.3      yaml_2.2.0         MASS_7.3-51.4
## [37] plyr_1.8.4         recipes_0.1.6      grid_3.5.3
## [40] crayon_1.3.4       haven_2.1.1        splines_3.5.3
## [43] hms_0.5.1          zeallot_0.1.0      pillar_1.4.2
## [46] reshape2_1.4.3     codetools_0.2-16   stats4_3.5.3
## [49] glue_1.3.1         evaluate_0.14      modelr_0.1.5
## [52] vctrs_0.2.0        foreach_1.4.7      cellranger_1.1.0
## [55] gtable_0.3.0       assertthat_0.2.1   xfun_0.9
## [58] gower_0.2.1        prodlim_2018.04.18 broom_0.5.2
## [61] class_7.3-15       survival_2.44-1.1  viridisLite_0.3.0
## [64] timeDate_3043.102  iterators_1.0.12   lava_1.6.6
## [67] ipred_0.9-9
```