Harvardx PH259 Capstone

Eden Axelrad

2022-06-27

SECTION 1: Setup

In this section we set up the initial data set from MovieLens. The data for the MovieLens 10M dataset can be found here:

- https://grouplens.org/datasets/movielens/10m/
- http://files.grouplens.org/datasets/movielens/ml-10m.zip

```
# Download the zipped file, unzip, and load both ratings and movies data
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")

# Clean up the classes for the movies data frame
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(movieId),
         title = as.character(title),
         genres = as.character(genres))

# Left join the ratings with movies using their movie ID
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

# Remove extraneous data from environment
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

SECTION 2: Introduction and Overview

The MovieLens recommendation site was launched in 1997 by GroupLens Research (which is part of the University of Minnesota). Today, the MovieLens database is widely used for research and education purposes. In total, there are approximately 11 million ratings and 8,500 movies. Each movie is rated by a user on a scale from 1/2 star up to 5 stars.

The goal of this project is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. The key steps that are performed include:

- Perform exploratory analysis on the data set in order to identify valuable variables
- Generate a naive model to define a baseline RMSE and reference point for additional methods
- Generate linear models using average movie ratings (movie effects) and average user ratings (user effects)
- Utilize matrix factorization to achieve an RMSE below the desired threshold
- Present results and report conclusions

SECTION 3: Methods / Analysis

This section explains the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and modeling approach

SECTION 3.1: Exploratory Analysis

```
# Set seed to 92 and generate a random sample of 100,000 observations from edx
options(dplyr.summarise.inform = FALSE)
set.seed(92, sample.kind = "Rounding")
exp_edx <- edx[(sample(nrow(edx), size = 100000)), ]

# Clean up the data set and add time based variables including year, movie age, etc.
exp_edx <- exp_edx %>%
  mutate(rating_date = lubridate::as_datetime(timestamp),
         rating_year = lubridate::year(rating_date),
         rating_month = lubridate::month(rating_date),
         release_year = as.double(
           gsub("[\\(\\)]", "", regmatches(title, gregexpr("\\(.*?\\)", title))[[1]])),
         rating_gap = rating_year-release_year,
         movie_age = 2022-release_year)
head(exp_edx)
```

```
##    userId movieId rating  timestamp                          title
## 1:   5296     882    2.0  974614347     Trigger Effect, The (1996)
## 2:  46020    1307    3.5 1178570445 When Harry Met Sally... (1989)
## 3:  22401    1961    4.0  974750310               Rain Man (1988)
## 4:  46071    5349    2.5 1053815040             Spider-Man (2002)
## 5:  59204     230    3.0  843164666       Dolores Claiborne (1995)
## 6:  59845     916    4.0  940097535           Roman Holiday (1953)
##                                 genres         rating_date rating_year
## 1:                      Drama|Thriller 2000-11-19 06:12:27        2000
## 2:                      Comedy|Romance 2007-05-07 20:40:45        2007
## 3:                               Drama 2000-11-20 19:58:30        2000
## 4: Action|Adventure|Sci-Fi|Thriller 2003-05-24 22:24:00        2003
## 5:                      Drama|Thriller 1996-09-19 20:24:26        1996
## 6:                      Comedy|Romance 1999-10-16 18:12:15        1999
##    rating_month release_year rating_gap movie_age
## 1:           11         1996          4        26
## 2:            5         1996         11        26
## 3:           11         1996          4        26
## 4:            5         1996          7        26
## 5:            9         1996          0        26
## 6:           10         1996          3        26
```

```
# Create summary table for average rating by movie (movieId)
movie_add <- exp_edx %>%
  group_by(movieId) %>%
  summarize(n_ratings = n(),
            avg_movie_rating = mean(rating))
head(movie_add)
```

```
## # A tibble: 6 x 3
##   movieId n_ratings avg_movie_rating
##     <dbl>     <int>            <dbl>
## 1       1       282             3.91
## 2       2       109             3.17
## 3       3        78             3.08
## 4       4        18             2.75
## 5       5        62             3.09
## 6       6       122             3.72
```

```
# Create a summary for average rating by user (userId)
user_add <- exp_edx %>%
  group_by(userId) %>%
  summarize(avg_user_rating = mean(rating))
head(user_add)
```

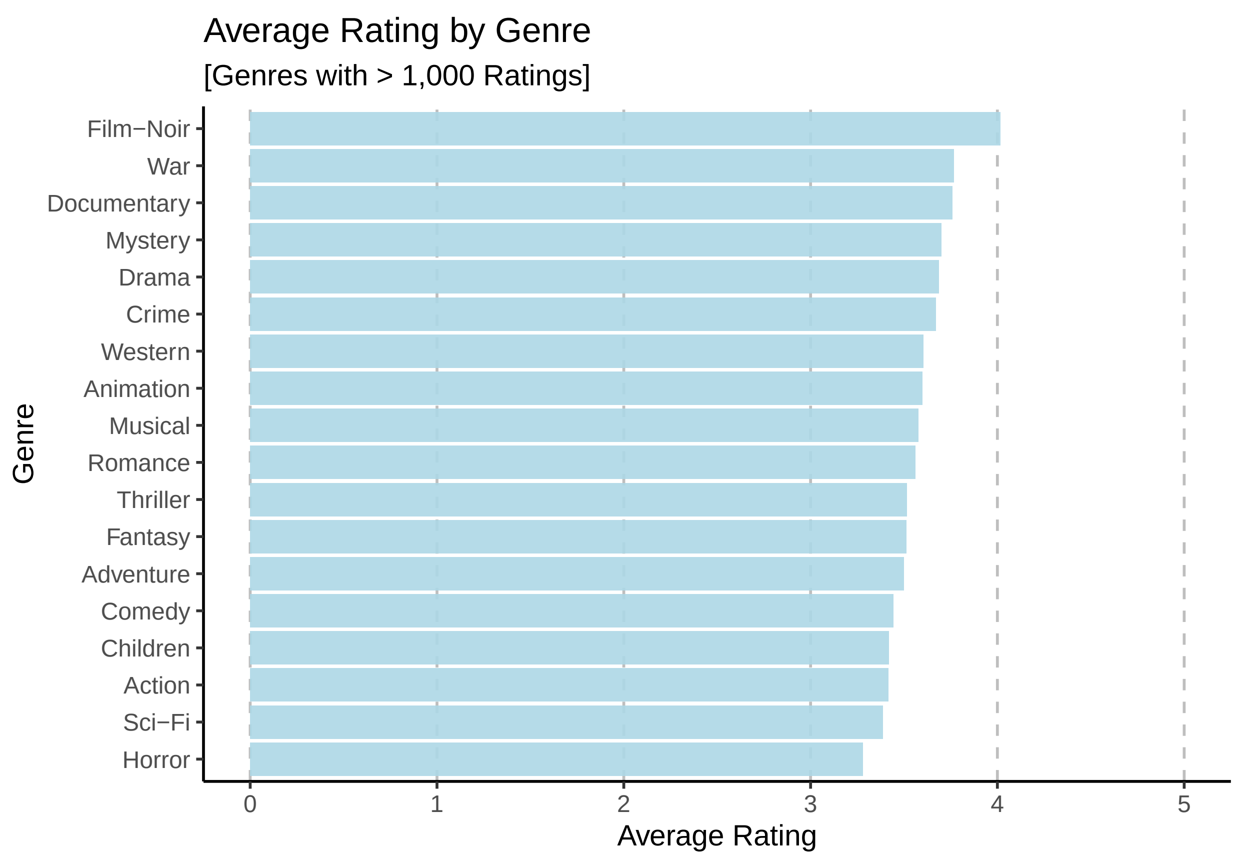
```
## # A tibble: 6 x 2
##   userId avg_user_rating
##    <int>           <dbl>
## 1      8            3.36
## 2     10            5
## 3     11            3
## 4     12            4
## 5     13            3
## 6     14            5
```

```r
# Find average rating by individual genre and genre combinations
genre_list <- exp_edx %>%
  mutate(genre_rating = strsplit(genres, "|", fixed = TRUE)) %>%
  as_tibble() %>%
  select(rating, genre_rating) %>%
  unnest(genre_rating) %>%
  group_by(genre_rating) %>%
  summarize(avg_genre_rating = mean(rating), n = n())

genre_add <- cbind.data.frame(id = 1:length(unique(exp_edx$genres)),
                              genres = unique(exp_edx$genres)) %>%
  mutate(genre_rating = strsplit(genres, "|", fixed = TRUE)) %>%
  unnest(genre_rating) %>%
  inner_join(genre_list, by = "genre_rating") %>%
  group_by(id, genres) %>%
  summarize(avg_genre_rating = mean(avg_genre_rating))
head(genre_add)
```

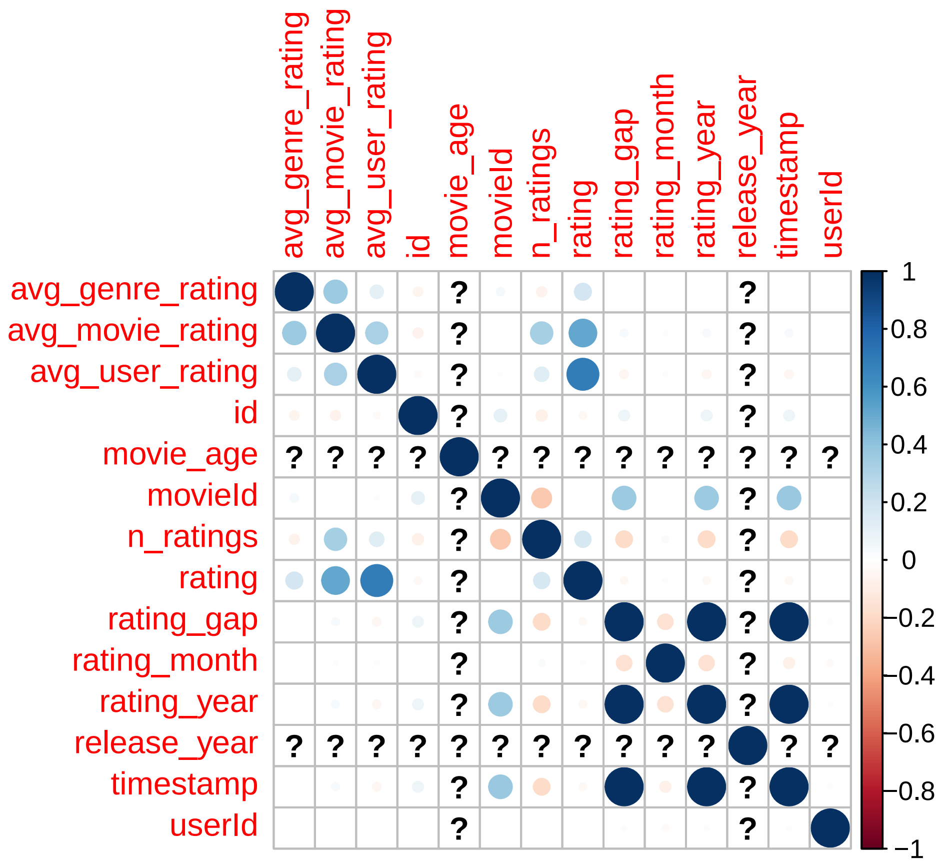
```
## # A tibble: 6 x 3
## # Groups:   id [6]
##      id genres                             avg_genre_rating
##   <int> <chr>                                         <dbl>
## 1     1 Drama|Thriller                                 3.60
## 2     2 Comedy|Romance                                 3.50
## 3     3 Drama                                          3.69
## 4     4 Action|Adventure|Sci-Fi|Thriller               3.45
## 5     5 Comedy                                         3.44
## 6     6 Crime|Drama|Romance|Thriller                   3.61
```

```r
# Showcase average rating by genre
genre_list %>%
  filter(n > 1000) %>%
  mutate(genre_rating = fct_reorder(genre_rating, avg_genre_rating)) %>%
  ggplot(aes(x = avg_genre_rating, y = genre_rating)) +
  geom_col(fill = "lightblue", alpha = 0.9) +
  labs(y = "Genre",
       x = "Average Rating",
       title = "Average Rating by Genre",
       subtitle = "[Genres with > 1,000 Ratings]") +
  theme_classic() +
  scale_x_continuous(limits = c(0, 5),
                     breaks = 0:5,
                     labels = 0:5) +
  theme(panel.grid.major.x = element_line(linetype = "dashed", color = "gray"))
```



```r
# Combine all new fields
exp_edx <- left_join(exp_edx, movie_add, by = "movieId")
exp_edx <- left_join(exp_edx, user_add, by = "userId")
exp_edx <- left_join(exp_edx, genre_add, by = "genres")

# View correlation plot for all numeric variables in the exploratory data set
corrplot::corrplot(exp_edx %>% select_if(., is.numeric) %>% cor(),
                   method = 'circle', order = 'alphabet')
```



```r
# Remove all exploratory data from environment to free up space
rm(movie_add, user_add, genre_add, genre_list, exp_edx)
```

The correlation plot above shows us that movie ratings are most highly correlated with average user rating and average movie rating. There is also a small, if not negligible association with number of ratings and genre as well. For the purposes of the actual modelling, the two largest effects (movie bias and user bias) are explored.

SECTION 3.2: Actual Methods

Partition the edx data in to a test set with 20% and train set with 80%. Use the train and test sets to develop various models and assess their RMSE. Begin with a naive model and incrementally add variables, regularization, and factorization to meet RMSE threshold.

```r
# Partition the data in to a test set with 20% and train set with 80%
# Set seed to 92 for reproducing results
set.seed(92, sample.kind = "Rounding")
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
test_set <- edx %>% slice(test_index)
train_set <- edx %>% slice(-test_index)

# Optional: remove edx data set to free up space
rm(test_index, edx)
```

Method #1: Naive Model

```
### Method #1: Naive Model
# Start off with a naive model that uses the average rating to predict movie ratings
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(pred = mu_hat,
                   obs = test_set$rating)

naive_rmse
```

```
## [1] 1.060362
```

Method #2: Mean + Average Movie Rating

```
### Method #2: Mean + Average Movie Rating
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

pred_bi <- mu_hat + test_set %>%
  left_join(bi, by = "movieId") %>%
  pull(b_i)

model1_rmse <- RMSE(pred = pred_bi,
                    obs = test_set$rating,
                    na.rm = TRUE)
rm(pred_bi)

model1_rmse
```

```
## [1] 0.9441401
```

Method #3: Mean + Average Movie Rating + Average User Rating

```
# Method #3: Mean + Average Movie Rating + Average User Rating
bu <- train_set %>%
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

pred_bu <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(pred_bu = mu_hat + b_i + b_u) %>%
  pull(pred_bu)

model2_rmse <- RMSE(pred = pred_bu,
                    obs = test_set$rating,
                    na.rm = TRUE)
rm(pred_bu)

model2_rmse
```

```
## [1] 0.8668655
```

Method #4: Matrix factorization

Because linear models alone did not provide a sufficient RMSE, further methods are applied. A common approach used in recommender systems (think Netflix or YouTube) is matrix factorization. The basic premise of matrix factorization is that it identifies features that explain the interactions between users and items (in our case movies). It then leverages these associations (latent features) to make predictions.

To perform the matrix factorization I followed the general process described in the recosystem vignette. The vignette describes 5 main steps but I was able to achieve the desired RMSE with only the required (3) steps and default parameters. Thus, no tuning was necessary for this project. The non-optional steps are:

1. Create a reference class object using Reco()
2. Train the model using $train()
3. Use $predict() to compute predicted values

Additionally, the default parameters are as follows:

- 10 latent factors
- Regularization parameters for item factors from 0 (L1) to 0.1 (L2)
- Regularization parameters for user factors from 0 (L1) to 0.1 (L2)
- 20 iterations
- 20 bins

```
# Convert the train and test sets into recosystem input format
set.seed(92, sample.kind = "Rounding")
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating,
                                           date = date))

test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating,
                                           date = date))

# Step 1. Create a reference class object using Reco()
r <-  Reco()

# Step 2. Train the algorithm
r$train(train_data)
```

```
## iter      tr_rmse          obj
##    0       0.9727   1.1962e+07
##    1       0.8832   1.0674e+07
##    2       0.8700   1.0597e+07
##    3       0.8513   1.0406e+07
##    4       0.8430   1.0332e+07
##    5       0.8369   1.0290e+07
##    6       0.8318   1.0254e+07
##    7       0.8283   1.0226e+07
##    8       0.8259   1.0210e+07
##    9       0.8241   1.0193e+07
##   10       0.8230   1.0187e+07
##   11       0.8220   1.0177e+07
##   12       0.8213   1.0173e+07
##   13       0.8206   1.0168e+07
##   14       0.8201   1.0162e+07
##   15       0.8196   1.0160e+07
##   16       0.8193   1.0158e+07
##   17       0.8189   1.0154e+07
##   18       0.8187   1.0153e+07
##   19       0.8185   1.0152e+07
```

```
# Step 3. Calculate the predicted values (with $predict()) using Reco test_data
# Directly return R vector
pred_MtrxFct <-  r$predict(test_data, out_memory())

# Find RMSE for matrix factorization predicted output
MtrxFct_rmse <- RMSE(test_set$rating, pred_MtrxFct)

MtrxFct_rmse
```

```
## [1] 0.8346059
```

SECTION 3.3: Validation

```
# Since the matrix factorization gets us where we need to be,
# repeat the above process but with our validation set

# Convert the validation set into recosystem input format
validation_data  <-  with(validation,  data_memory(user_index = userId,
                                                   item_index = movieId,
                                                   rating = rating,
                                                   date = date))

# Calculate the predicted values using Reco validation_data
pred_MtrxFct_Val <- r$predict(validation_data, out_memory())

# Calculate RMSE for the predicted values using matrix factorization
MtrxFct_rmse_val <- RMSE(validation$rating, pred_MtrxFct_Val)

MtrxFct_rmse_val
```

```
## [1] 0.8339622
```

SECTION 4: Results

Linear models did not provide sufficiently low RMSEs, however matrix factorization did achieve the desired RMSE threshold of < 0.86490. The final test set RMSE was 0.83461 and the final validation set RMSE was 0.83396. Results are presented in the summary table below.

```r
# This section presents the modeling results
# Final results table with test on validation set
results_tbl <- tibble(
  Method =
    c("Method #1", "Method #2", "Method #3", "Method #4", "Final Validation"),
  Model =
    c("Naive Model", "Mean + Movie", "Mean + Movie + User",
      "Matrix Factorization", "Matrix Factorization"),
  RMSE = c(naive_rmse, model1_rmse, model2_rmse, MtrxFct_rmse, MtrxFct_rmse_val)) %>%
  mutate(`Estimated Points` = case_when(
    RMSE >= 0.90000 ~ 5,
    RMSE >= 0.86550 & RMSE <= 0.89999 ~ 10,
    RMSE >= 0.86500 & RMSE <= 0.86549 ~ 15,
    RMSE >= 0.86490 & RMSE <= 0.86499 ~ 20,
    RMSE < 0.86490 ~ 25)
    )

results_tbl %>%
  mutate(RMSE = round(RMSE, digits = 5)) %>%
  knitr::kable()
```

Method	Model	RMSE	Estimated Points
Method #1	Naive Model	1.06036	5
Method #2	Mean + Movie	0.94414	5
Method #3	Mean + Movie + User	0.86687	10
Method #4	Matrix Factorization	0.83461	25
Final Validation	Matrix Factorization	0.83396	25

SECTION 5: Conclusion

This Capstone project utilized the MovieLens data set to train and test multiple models and approaches for recommendor systems. This project showcases the effects of movies and users on linear models. It also showcases the strength of matrix factorization in reducing RMSE. The desired RMSE threshold of $<$ 0.86490 was surpassed, and a final validation RMSE of 0.83396 was achieved. For future work, the the matrix factorization model can be further refined with tuning parameters. This would result in an optimized model and (likely) lower RMSE.