PH125.9x MovieLens submission

Lucas Laursen

2022-11-02

1. Introduction

There is a special pleasure in browsing all the movies at the video rental place, for those of old enough to remember them, and then going home without having chosen anything. You might encounter strange Westerns set outside the West or horror films too horrible to watch and by the time you had gone through all the options, it was too late to start watching anything. But for days when you actually want to watch a new film, and anytime we use our brave new streaming options, bereft of a helpful clerk, we need recommendations.

As we learned in earlier classes of this data science certificate, we can model collective preferences and use the model to can make recommendations. So, in this class, I used a movie rating dataset to practice machine learning methods and build a recommendation model. Here's how I did it, with handy code so graders and other interested parties can follow along. I used R version 4.0.1.

First, I downloaded the data following the class instructions and divided it into a training portion and a validation portion. Again following the class code, I applied some modeling methods and noted how well they fit the validation data. Then, to improve the model's performance, I added additional modeling until achieving an RMSE $<$ 0.86490. Finally, I commented on the meaning of my model and the limitations I knew of.

1.1 Data wrangling

First, let's make sure R has the packages we'll need:

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
#lubridateis supposed to be part of the tidyverse, but didn't load properly on my machine...
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")
if(!require(magrittr)) install.packages("magrittr", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(lubridate)
library(caret)
library(data.table)
library(magrittr)
```

The data we'll use is the so-called MovieLens 10M dataset: https://grouplens.org/datasets/movielens/10m/
http://files.grouplens.org/datasets/movielens/ml-10m.zip

Note: this next portion of code could take a couple of minutes to run. It downloads the data from one of the above URLs, unzips it, parses the data from two files by their column names and combines the data into a single dataframe called movielens containing six columns and ten million rows.

```r
options(timeout=1000)
dl <- tempfile() #creates a string as a placeholder name for later use
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", copy = TRUE)

movielens <- mutate(movielens, date = as_datetime(timestamp))
movielens <- mutate(movielens, date = round_date(date, unit = "year"))
```

The validation set will be 10% of MovieLens data, using the same sort of random partitioning we learned earlier in the class.

```r
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,]
```

Then we'll make sure the userId and movieId in the validation set are also in the edx set. This will prevent awkward NAs when we do some dataframe joining later on. Ask me how I know!

```r
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

Next we add the rows removed from the validation set back into the edx set, remove temporary files, and write both sets to file for future use. We don't want to have to re-download the data every time we test our code.

```r
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

write_csv(edx, "edx.csv", )
write_csv(validation, "validation.csv")
```

That's great. Now, to be rigorous, we should re-divide the edx dataset into training and test sets for optimizing our model. Then, once we've decided on the best model we can retrain it using the full edx dataset and test it against the validation dataset. This avoids overtraining.

```r
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
```

```
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in 'test' set are also in 'train' set
test <- temp %>% semi_join(train, by = "movieId") %>% semi_join(train, by = "userId")

# Add rows removed from 'test' set back into 'train' set
removed <- anti_join(temp, test)
train <- rbind(train, removed)

rm(test_index, temp, removed)

write_csv(train, "train.csv")
write_csv(test, "test.csv")
```

While I was building this model, I found it easier to start from the local files to avoid re-downloading the data. Like the download, this also takes some time!

```
edx <- read.csv("edx.csv", sep = ",")
validation <- read.csv("validation.csv", sep = ",")
train <- read.csv("train.csv", sep = ",")
test <- read.csv("test.csv", sep = ",")
```

At this point, we have a training dataframe partition called 'train' that contains 8.1 million observations of 6 variables and a testing dataframe partition called 'test' with 899,990 independent observations of the same variables. We have a separate validation dataframe partition called 'validation' of 1 million observations that we won't touch until the end.

2. Methods/Analysis

My initial goal now is to select and build some machine learning models that train on the 'train' data and generate an RMS less than 0.86490 on the 'test' data. Then I'll apply that model to the full edx (train+test) dataset and test it against the validation dataset.

First, I'll define the function that will evaluate my model predictions against the test data and generate an RMSE.

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

2.1 Model 1

Now I will follow code from the textbook to test some simplistic models before trying to make improvements on my own initiative. A starting point is predicting that any given movie rating will be the mean of all known ratings:

```
  mu_hat <- mean(train$rating)
  naive_rmse <- RMSE(train$rating, mu_hat)
```

Which gets us a naive RMSE of 1.060, which is 22.6% over the target but let's store it anyway for comparison with subsequent models:

```
target_rmse <- 0.8649
rmse_results <- tibble(method = "Target RMSE", RMSE = num(target_rmse, sigfig=5), "% relative to RMSE" =
```

NB: Knitting the .Rmd file into LaTeX doesn't wrap lines of code and despite a few tries I couldn't figure it out. Sorry. The code you need to test the model is in the appendix at the bottom of this file and the full code for all the testing I did is in the separate file movielensLaursencode.R.

2.2 Model 2

Now, for a more complex model we could add the the least squares error, b_hat_i, of the average rating for each given movie to the overall average rating. This accounts for how some movies get higher ratings than others across the entire population.

We can find the least squares estimate b_hat_i because it is the average of the rating prediction Y_u,i minus the average rating mu_hat. As in the textbook, we will ignore the hat notation for the rest of this project.

```
  rm(mu_hat)
  mu <- mean(train$rating) #here we dispense with the _hat notation, following the textbook
  rm()
  movie_avgs <- train %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))
```

We can test predictions based on the least squares estimate for movies (b_i) against the validation dataset:

```
  predicted_ratings <- mu + test %>%
    left_join(movie_avgs, by='movieId') %>%
    pull(b_i)
  movie_rmse <-  RMSE(predicted_ratings, test$rating)
```

I get 0.94296, better than our original RMSE, but still 9% above the target. Let's store it and continue:

```
rmse_results <- rmse_results %>% add_row(method = "Mean + Movie Effect Model", RMSE = movie_rmse, "% rel
```

2.3 Model 3

The textbook says a next step would be to try to incorporate systematic variation between users, because some users just rate all movies higher or lower than other users. A straight-up linear model approach would be too slow, so again we resort to the least-mean-squares approach.

```
  user_avgs <- train %>%
  group_by(userId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_u = mean(rating - mu - b_i))
```

That step defined the b_u column to hold each user's average rating and stored it in the user_avgs dataframe. Now we can define our new, multi-part model, which incorporates overall average, movie effects, and superuser effects.

```
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_rmse <- RMSE(predicted_ratings, test$rating)
```

This is now down to 0.8647, 0.025% below the 0.8649 target!

```
rmse_results <- rmse_results %>% add_row(method = "Mean + Movie + User Effects Model", RMSE = user_rmse
```

It's so close that we might want to improve our model performance to leave a little breathing room, especially since we haven't yet tested it against the independent validation dataset.

What about superusers, who have rated more than 100 films? Can they improve our estimates?

2.4 Model 4

```
superuser_avgs <- train %>%
  group_by(userId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_v = mean(rating - mu - b_i))
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(superuser_avgs, by='userId') %>%
  mutate(pred = mu + b_i + coalesce(b_v,0.0)) %>%
  #coalesce() prevents NAs where test + superuser_avgs don't overlap
  pull(pred)
superuser_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Movie + SuperUser Model", RMSE = superuser_rms
```

0.86468. No improvement. We'll call that a dead end.

Moving on to consider genre as a source of variation in ratings:

2.5 Model 5

```
genres_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
genres_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Movie + User + Genre Model", RMSE = genres_rm
```

genres_rmse = 0.86432, which is 0.067% below the target. Things are improving but still too close for comfort.

2.6 Model 6

What about following one of the clues from the related quiz from the machine learning module of this certificate? Let's try filtering genres to ensure we only use the genre combinations with more than 1000 ratings:

```
genres_filt <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>% mutate(n_ratings=n()) %>% filter(n_ratings>=1000) %>%
  summarize(b_h = mean(rating - mu - b_i - b_u))

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_filt, by='genres') %>%
  mutate(pred = mu + b_i + b_u + coalesce(b_h,0.0)) %>%
  #coalesce() prevents NAs where test + superuser_avgs don't overlap
  pull(pred)
genres_filt_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Movie + User + Filtered Genre Model", RMSE = g
```

Now we're back up to 0.86436, which is higher than before! Let's not waste our time going any further down that path.

2.7 Model 7

Instead, what about regularization using a guessed initial lambda, as per the textbook?

```
lambda <- 3
mu <- mean(train$rating)
movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
movies_reg_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Regularized Movie Model", RMSE = movies_reg_r
```

movies_reg_rmse = 0.94294, which is 0.00005 better than without regularization.

2.8 Model 8

But what if I can optimize lambda through cross-validation?

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(train$rating)
just_the_sum <- train %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test$rating))
})
lambdas[which.min(rmses)]
movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambdas[which.min(rmses)]), n_i = n())
predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
movies_cv_reg_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Cross-Validated Regularized Movie Model", RMSE
```

0.94294, a tiny bit better than the movie effects RMSE without regularization of 0.94296.

2.9 Model 9

Ok, not that much better, but what if we combined movie & user effects & regularization? Maybe user effects are more volatile than movie effects and penalizing the extremes will offer more improvement. Judging from how slowly the preceding model ran this one will be slow, too…

```
regularization <- function(lambda, trainset, testset){
  mu <- mean(train$rating)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}

#for the following sequence of lambdas--THIS CODE IS SLOW
lambdas <- seq(0, 10, 0.25)
#apply the function defined above to generate the possible RMSEs
rmses <- sapply(lambdas,
                regularization,
                trainset = train,
                testset = test)
#and store the optimal lambda
lambda <- lambdas[which.min(rmses)]

#then use that to find the regularized RMSE for movie + user effects

movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
user_avgs <- train %>%
  group_by(userId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_u = mean(rating - mu - b_i))

movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- train %>%
  group_by(userId) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_cv_reg_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Cross-Validated Regularized Movie + User Model
```

0.8641 Ok, this is a tad better, below the threshold. What happens if we re-train the model on the full edx training dataset and test it against the independent validation dataset?

#2.10 Model 9 vs. Validation data

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas,
                regularization,
                trainset = edx,
                testset = validation)
```

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

movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- edx %>%
  group_by(userId) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
validated_user_cv_reg_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- rmse_results %>% add_row(method = "Winning model vs. validation data", RMSE = validated_
```

0.864817, just below the threshold!

2.11 Model 10

How about we incorporate genre, too, to see if we can improve this?

```
genre_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

regularization <- function(lambda, trainset, testset){
  mu <- mean(train$rating)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
  b_g <- trainset %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_u - b_i - mu)/(n()+lambda))
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    filter(!is.na(b_i), !is.na(b_u), !is.na(b_g)) %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, testset$rating))
}

lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas,
                regularization,
                trainset = train,
                testset = test)
lambda <- lambdas[which.min(rmses)]

movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- train %>%
  group_by(userId) %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  left_join(user_avgs, by='userId') %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
genre_reg_avgs <- train %>%
  group_by(genres) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by= 'movieId') %>%
  left_join(genre_avgs, by= 'genres') %>%
  summarize(b_g = sum(rating - mu - b_u)/(n()+lambda), n_u = n())
predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  left_join(genre_reg_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
genres_cv_reg_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Movie + User + Genre Cross-Validated Regulari
```

WEIRD: gone up to 0.9126… more than 5% over the target.

2.12 Model 11

One idea hinted at in the Module 8 quiz is time-dependency. So I go back to the top of this code and add a column to the very first dataframe (movielens) that takes the timestamp for each movie and rounds it to the nearest year for simplicity. Then we can check whether the year a movie was made affects its ratings.

```
#adapting code from the genre section to consider the average rating for each year
time_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))

#adapting the regularization function to include time...
  regularization <- function(lambda, trainset, testset){
  mu <- mean(train$rating)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
  b_t <- trainset %>%
    left_join(b_u, by="userId") %>%
    group_by(date) %>%
    summarize(b_t = sum(rating - b_u - b_i - mu)/(n()+lambda))
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "date") %>%
    filter(!is.na(b_i), !is.na(b_u), !is.na(b_t)) %>%
    mutate(pred = mu + b_i + b_u + b_t) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}

lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas,
                regularization,
                trainset = train,
                testset = test)
lambda <- lambdas[which.min(rmses)]

movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- train %>%
  group_by(userId) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
time_reg_avgs <- train %>%
  group_by(date) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  left_join(time_avgs, by= 'date') %>%
  summarize(b_t = sum(rating - mu - b_u)/(n()+lambda), n_u = n())
predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  left_join(time_reg_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  pull(pred)
time_cv_reg_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- rmse_results %>% add_row(method = "Mean + Cross-Validated Regularized Movie + User + Ti
```

0.86557, 0.0776% OVER the target. Sigh.

Maybe instead of year effect we need to look at day of the week, as hinted at in a related quiz in the machine learning class? At any rate, I'm going to quite while I'm ahead, even if it's only a little ahead.

3. Results

The results of my modeling:

```
rmse_results
```

```
# A tibble: 12 × 3
   method                                                           RMSE ‘% relative to RMSE‘
   <chr>                                                       <num:5>              <num:3>
 1 Target RMSE                                                 0.8649                0
 2 Just the average                                            1.0604               22.6
 3 Mean + Movie Effect Model                                   0.94296               9.03
 4 Mean + Movie + User Effects Model                           0.86468              -0.0249
 5 Mean + Movie + SuperUser Model                              0.86468              -0.0249
 6 Mean + Movie + User + Genre Model                           0.86432              -0.0666
 7 Mean + Movie + User + Filtered Genre Model                  0.86436              -0.0626
 8 Mean + Regularized Movie Model                              0.94295               9.02
 9 Mean + Cross-Validated Regularized Movie Model              0.94294               9.02
10 Mean + Cross-Validated Regularized Movie + User Model       0.86414              -0.0883
11 Winning model vs. validation data                           0.86482              -0.00960
12 Mean + Movie + User + Genre Cross-Validated Regularized Model 0.91260             5.51
13 Mean + Cross-Validated Regularized Movie + User + Time Model  0.86590             0.115
```

4. Conclusion

So the ‘winning’ model that just barely beats the target on the validation dataset is model 9, the cross-validated, regularized model that incorporates the mean rating, movie effects, and user effects. This implies that some movies really are better (or worse) for most audiences and the inverse, some people generally like movies more or less than other people do. Genre and the year of the movie had smaller effects and seem to have made the overall model worse.

A limitation of this model is that it only barely gets below the target RMSE, so there is still room for improvement.

I am also not able to explain why the film and genres were less important–only that they don't seem to have improved the model.

Maybe this model can replace the video store clerk now that we're all using streaming services but it offers only limited insight into why we're going to like a particular movie. That would be a great challenge for a future project.

Appendix: top-to-bottom, uncommented code for just the ‘winning’ model, for easy verification.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", 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")
if(!require(magrittr)) install.packages("magrittr", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(lubridate)
library(caret)
library(data.table)
library(magrittr)

options(timeout=1000)
dl <- tempfile() #creates a string as a placeholder name for later use
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", copy = TRUE)

movielens <- mutate(movielens, date = as_datetime(timestamp))
movielens <- mutate(movielens, date = round_date(date, unit = "year"))

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,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

write_csv(edx, "edx.csv", )
write_csv(validation, "validation.csv")

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

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

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

rm(test_index, temp, removed)

write_csv(train, "train.csv", )
write_csv(test, "test.csv")

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

target_rmse <- 0.8649
mu <- mean(train$rating)
naive_rmse <- RMSE(train$rating, mu)

movie_avgs <- train %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))

user_avgs <- train %>%
  group_by(userId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_u = mean(rating - mu - b_i))

regularization <- function(lambda, trainset, testset){
  mu <- mean(train$rating)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}


lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas,
                regularization,
                trainset = train,
                testset = test)
lambda <- lambdas[which.min(rmses)]
movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- train %>%
  group_by(userId) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_cv_reg_rmse <- RMSE(predicted_ratings, test$rating)


lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas,
                regularization,
                trainset = edx,
                testset = validation)
lambda <- lambdas[which.min(rmses)]
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  left_join(movie_avgs, by='movieId') %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
user_reg_avgs <- edx %>%
  group_by(userId) %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_reg_avgs, by= 'movieId') %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())
predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
validated_user_cv_reg_rmse <- RMSE(predicted_ratings, validation$rating)

#artificial line breaks...you'll need to delete them if you paste this code
rmse_results
<- tibble(method = "Target RMSE", RMSE = num(target_rmse, sigfig=5), "% relative to RMSE" = num((target_
add_row(method = "Just the average", RMSE = naive_rmse, "% relative to RMSE" =
  num((naive_rmse - target_rmse)/(target_rmse)*100, sigfig=3)) %>%
add_row(method = "Model", RMSE = user_cv_reg_rmse, "% relative to RMSE" =
  num((user_cv_reg_rmse - target_rmse)/(target_rmse)*100, sigfig=3)) %>%
add_row(method = "Above model vs. validation data", RMSE = validated_user_cv_reg_rmse,
  "% relative to RMSE" = num((validated_user_cv_reg_rmse - target_rmse)/(target_rmse)*100, sigfig=3))

rmse_results
```