MovieLens Project

Elizabeth

2023-08-30

Introduction

This dataset is based on the MovieLens data, a dataset that is comprised of many user submissions to different movies. The data includes the timestamp, title, year, genre, movie ID, user ID, and the rating.

The purpose of this analysis is to determine the likelihood of predicting ratings accounting for user bias, average rating per movie, overall mean of movie ratings, and the day that the rating was placed.

Key steps include creating test and training sets, using a motivation-based regularization algorithm, and using the Root-Mean-Square-Error(RMSE) to calculate the final result.

Methods/Analysis

The first step was to create "edx" and "final_holdout_test" sets from the MovieLens datasets. This was done using the following code.

This code downloads the MovieLens dataset, creates a test set with 10% of the data, makes sure that userId and movieId in final hold-out set are also in the edx set, adds rows removed from the holdout test into the edx set, and then removes unnecessary variables.

```
library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat" #if the file doesn't exist, download it
if(!file.exists(ratings_file)) #downloads ratings
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file)) #downloads movies
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
```

```
                          stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId") #makes sure that all columns in ratings are in

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,] #remaining 90% of MovieLens data
temp <- movielens[test_index,] #temporary name for final dataset

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

This code creates a function that calculates the RMSE given known ratings and predicted ratings. The RMSE is the Root of the Mean Squared Error, which calculates the probability that an error will occur. A lower RMSE means a lower chance of error.

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

The beginning code used in the project is below. The variable "timestamp" was changed because a "day" variable is much easier to use functions on than a "datetime" variable.

```
library(tidyverse)
library(caret)
library(lubridate)
library(gam)

#add a "date" column to the edx function. Format dates and round them to the day
edx <- mutate(edx, date = as_datetime(timestamp))
#remove the times from the dates and format it Year-Month-Date
edx <- mutate(edx, date = round_date(date, unit = "day"))
```

Once we have the edx dataset properly formatted, we make the test and training sets for the model. The purpose of this model is to predict a user's rating for a film given the date. The test set is 20% of the edx data.

```
#create the train and test sets using indexes
set.seed(468)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE) # create a 20% test
test_set <- edx[test_index,]
train_set <- edx[-test_index,]

#make sure that test set doesn't include users, dates, and movies that don't appear in training
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "date")
```

d_ui is equal to the day of rating u of movie i. b_i is the average rating for movie i, b_u is the user-specific effect, and mu is the average rating.

The model used is a motivation-based regularization algorithm, which uses a penalty term to decrease the importance of estimates that have a small sample size. The penalty term is called lambda. The regularization algorithm decreases the chances for entries with large b_i (ex. unpopular and niche movies) to be overrepresented in the database.

This code finds the lamda that minimizes the RMSE for the model.

```
lambdas <- lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l) {
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  d_ui <- train_set %>% left_join(b_u, by=c("userId")) %>%
    group_by(date) %>%
    summarize(d_ui = mean(rating)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(d_ui, by = "date") %>%
    mutate(pred = (mu + b_i + b_u + d_ui)) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
lambdas[which.min(rmses)]
```

The minimum lambda is 4.75. Next, we run the interior code with the minimum lambda to find the predicted ratings for that lambda without applying a sequence to the entire section of code. This will give the predicted ratings for the lambda, which guarantees the lowest RMSE.

```
l <- 4.75
mu <- mean(train_set$rating)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))
d_ui <- train_set %>% left_join(b_u, by=c("userId")) %>%
  group_by(date) %>%
  summarize(d_ui = mean(rating)/(n()+l))
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(d_ui, by = "date") %>%
  mutate(pred = (mu + b_i + b_u + d_ui)) %>%
  pull(pred)
```

Results

The results are found by finding the RMSE of the predicted ratings with the minimizing lambda and the ratings of the final_holdout_test. You can find this with the code below:

```
RMSE(predicted_ratings, final_holdout_test$rating)
```

The RMSE for the final holdout test was 1.222359. This RMSE means that the model is very likely to be incorrect when using it for predictions. Possible causes for this high RMSE are overtraining (using test data during model training) or something missing from the model equation found in mutate() under predicted_ratings.

Conclusion

The dataset was put through a model that aims to decrease the representation of niche data terms that have high average ratings.

This model is not predicted to be very useful, as the RMSE is very high. Its limitations include a high amount of time taken to find the lowest lambda.