MovieLens

L. Schoch

2/11/2019

Overview

This project is an assignment for the Harvard Edx online course, **PH125.9x Data Science: Capstone**. The task is to create a movie recommendation system by developing and training a machine learning algorithm using the MovieLens 10M dataset (available at https://grouplens.org/datasets/movielens/10m). The MovieLens 10M dataset contains 10,000,054 ratings of 10,681 movies by 71,567 users. For this project, the data are split such that 90% of the observations are used as the training set, named edx, with the remaining 10% taken as the test set, named validation.

Accuracy was initially assigned to be used for evaluation of results. Considerable time and effort were devoted, unsuccessfully, to developing a model that achieved acceptable results using this metric. Later, residual mean squared error (rmse) was added as the preferred option for results evaluation and this is the metric used for this report.

The method outlined in the course was based on user and movie effects with regularization and this was the starting point for the results reported herein. Improvement was achieved by adding a genre effect. In addition, variables arising from this effects model, as well as other variables created from the existing data, were used as inputs to train an Rborist model from the caret package.

Data Exploration

Code for creation of the training set, edx, and the test set, validation, was provided to the student and it can be found in the separately submitted R script file. Edx consists of 9000055 observations of 6 variables including userId, movieId, rating, timestamp, title and genres. Ratings were provided by 69878 users.

As a first step in data exploration it was determined that there were no missing values in either the edx or validation dataset:

```
sum(is.na(edx))
```

```
## [1] 0
```

```
sum(is.na(validation))
```

```
## [1] 0
```

Proceeding with data exploration, Figure 1 shows the distribution of ratings in the edx dataset. The most common rating is 4 followed closely by 3. Not surprisingly, perhaps, there appears to be a predilection for whole numbered ratings as opposed to ratings in half-step increments.

Figure 2 shows the distribution of the number of ratings per movie. If we use the number of ratings as a proxy for the popularity of a movie, we can see that a few of the movies are wildly successful, garnering tens of thousands of ratings, and that the number of movies with a given popularity declines rapidly as popularity increases.

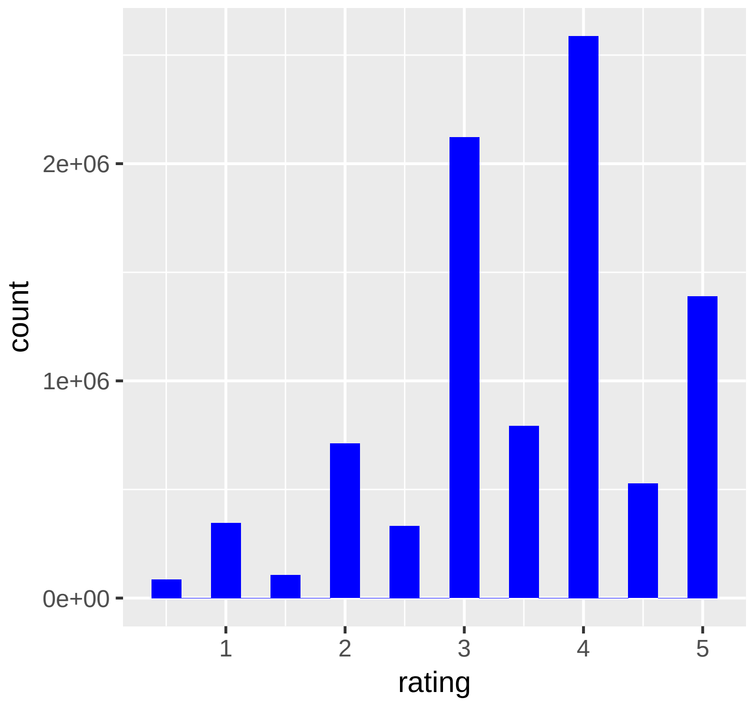

Figure 1: Distribution of ratings in edx.

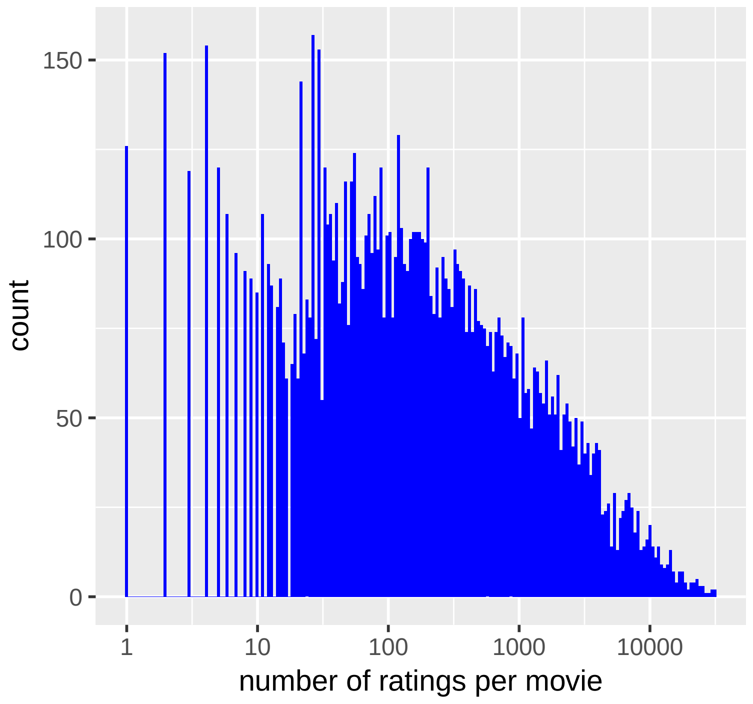

Figure 2: Distribution of ratings per movie in edx.

In Figure 3 observations can be made regarding the number of ratings per user. As expected, many users provided a relatively small number of ratings. To be more specific, we can use the `quantile` function to establish that 90% of users rated 301 or fewer movies each:

```r
users <- edx %>% group_by(userId) %>% summarize(n=n())
quantile(users$n, probs = 0.9)
```

```
## 90%
## 301
```

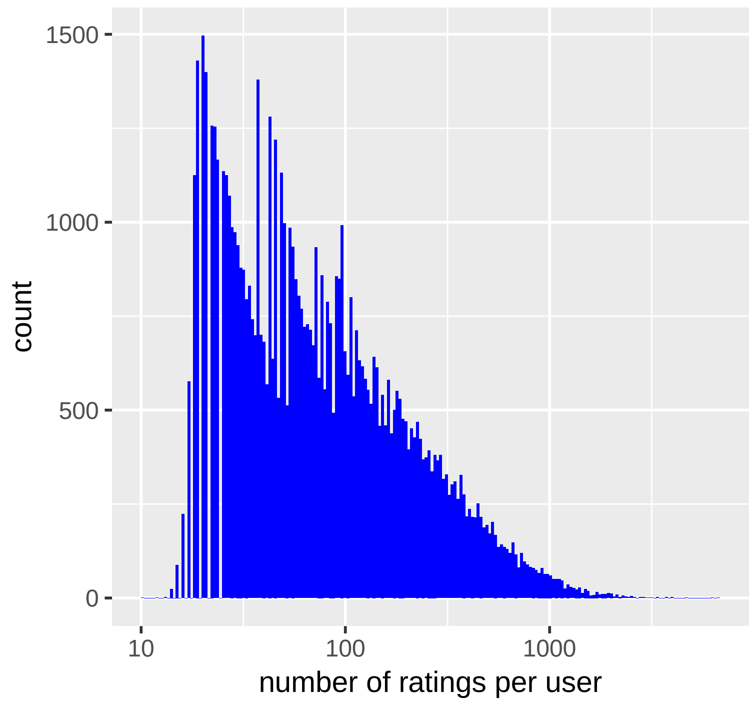

Figure 3: Distribution of ratings per user in edx.

And finally, R's empirical cumulative distribution function (`ecdf`) is used in Figure 4 to show, in a different way, that the bulk of ratings are provided by many users each rating relatively few movies.

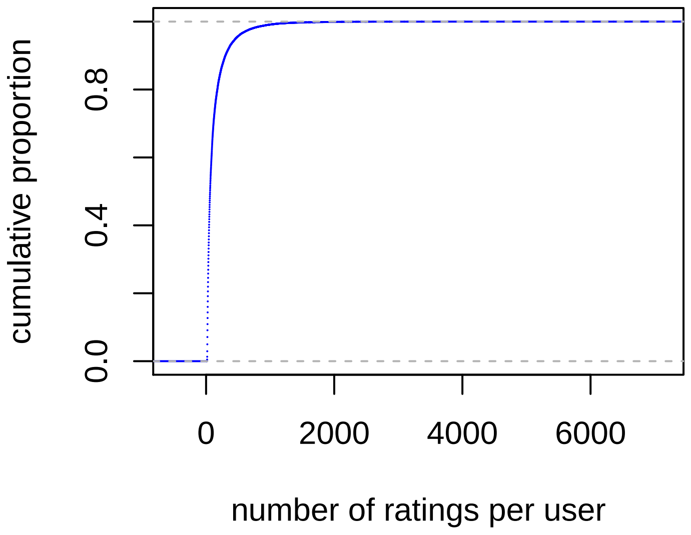

Figure 4: Cumulative distribution of ratings per user.

Methods

Effects Model

An effects model was used for this project:

$$y_{u,i,g} = \mu + b_u + b_i + b_g + \epsilon_{u,i,g}$$

where $y_{u,i,g}$ is the predicted rating for user u, movie i and genre g. μ is the average rating for all observations. b_u, b_i, and b_g are user, movie, and genre effects respectively and $\epsilon_{u,i,g}$ is the random error. Implementation of this model in R, including regularization, is detailed in the following code:

```r
# Create a function that computes the residual mean squared error (RMSE) for vectors of
# ratings and their corresponding predictors:
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

# Partition edx into a training set and a test set
set.seed(2019)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]
# Make sure userId and movieId in test_set are also in train_set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
# Add rows removed from test_set back into train_set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
# Remove objects no longer needed
rm(temp, test_index, removed)

# Calculate mu, mean rating of the training set, to be used in the prediction formula
mu <- mean(train_set$rating)

# Optimize the regularization tuning parameter, lambda
# Calculate rmses over a range of lambdas and find the lambda that minimizes rmse
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l)) # b_i = movie effect
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l)) # b_u = user effect
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - b_i - b_u - mu)/(n()+l)) # b_g = genre effect
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    .$pred
  # calculate RMSE
  return(RMSE(predicted_ratings, test_set$rating))
})

# determine optimum lambda that minimizes the rmses
optimum_lambda <- lambdas[which.min(rmses)]

# Calculate final model using optimum_lambda
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+optimum_lambda))
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+optimum_lambda))
b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - b_i - b_u - mu)/(n()+optimum_lambda))
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred
```

Rborist Model

Three variables arose from implementation of the effects model: b__u, b__i, and b__g, and three additional variables were created from the existing data: number of ratings per user, rating year and movie year. These 6 variables were used with the caret packge to calculate an Rborist model. Training was performed on a subset of the data - 500,000 randomly selected rows - to decrease computation times. Preprocessing was performed, centering each variable by subtracting the column mean, and the two tuning parameters, predFixed (randomly selected predictors) and minNode (minimal node size), were optimized. In the interest of decreasing computation time still further, optimization was performed using a reduced ntree (number of trees) setting of 50. After optimization, the final model was calculated using the optimized tuning parameters and the default Ntree setting of 500.

Code for the Rborist model:

```
# create new object from edx, adding b_i and b_u and b_g as columns
x <- edx %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres")
```

```r
# add a column for number of ratings per user
temp <- x %>% group_by(userId) %>% summarise(no.usr.ratings = n()) %>% ungroup()
x <- x %>% left_join(temp, by = "userId")
rm(temp)

# add columns for rating year and movie year
x <- x %>%
  mutate(rating_yr = as.numeric(format(round_date(as_datetime(timestamp), "day"), "%Y")),
         movie_yr = as.numeric(str_extract(title, "\\d{4}")))

# take a random sample of the data to decrease computation time
N <- 500000 # sample size
set.seed(2019)
x <- x[sample(1:nrow(x), N, replace = FALSE), ]

# create objects to be used for training the model
y <- x$rating
x <- x %>% select(b_i, b_u, b_g, no.usr.ratings, rating_yr, movie_yr)

# calculate Rborist model with preprocessing and parameter tuning
# use ntree = 50 to save computation time during parameter optimization
set.seed(2019)
grid <- expand.grid(predFixed = seq(1, 6), minNode = seq(50, 150, 25))
train_rf <- train(x, y,
                  method = "Rborist",
                  preProcess = "center",
                  trControl=trainControl(method = 'cv', number=5, p = 0.8),
                  nTree = 50,
                  tuneGrid = grid)

#calculate final model with ntree = 500 (default) and bestTune parameters
train_rf_final <- train(x, y,
                  method = "Rborist",
                  preProcess = "center",
                  trControl=trainControl(method = 'cv', number=5, p = 0.8),
                  nTree = 500,
                  tuneGrid = data.frame(predFixed = train_rf$bestTune$predFixed,
                                        minNode = train_rf$bestTune$minNode))
```

Results

Effects Model

The effects model was tested on the validation dataset and the results were evaluated using the residual mean squared error function. First, the three effects variables were joined to the validation data and then the predicted ratings were calculated by adding *mu* (the average rating for all observations in the training data) to the sum of the three effects variables (b_u + b_i + b_g) for each observation. The following code was used to calculate the RMSE for the effects model:

```r
# using the effects model to predict ratings for the validation set
# modify the validation set to include the three effects variables
validation_effects <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres")

# calculate the predicted ratings for the effects model
validation_effects_predict <- validation_effects %>%
  mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

# calculate the RMSE for the effects model
validation_effects_rmse <- RMSE(validation_effects_predict, validation$rating)
```

In the code above, `validation_effects_predict` is the vector of predicted ratings for the validation dataset using the effects model. The RMSE for the effects model, stored in `validation_effects_rmse`, is 0.86483.

Rborist Model

Figure 5 shows results of tuning parameter optimization for the Rborist model. The optimal parameters, predFixed = 3 and minNode = 150, were used for calculation of the final model.

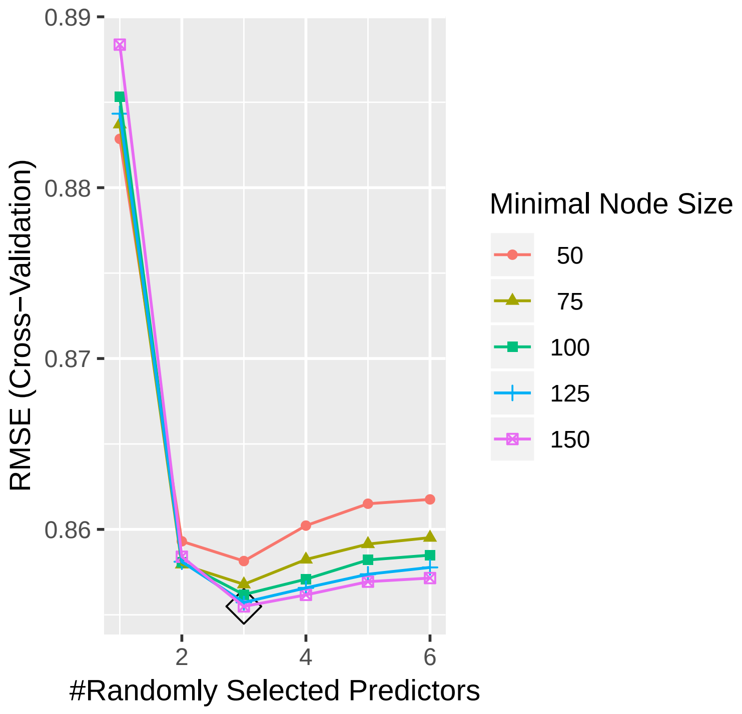

Figure 5: Tuning parameter optimization for the Rborist model

Similar to the effects model, evaluation of the Rborist model required modification of the validation dataset to include the variables that were used for training. Columns were added for b_u, b_i, b_g, ratings per user, movie year and rating year. These six columns were selected for fitting the Rborist model and results were once again evaluated using the RMSE function. The code:

```r
# using the rborist model to predict ratings for the validation set
# add the three effects variables to the validation dataset
validation_rborist <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres")

# add the column for number of ratings per user
temp <- validation_rborist %>% group_by(userId) %>% summarise(no.usr.ratings = n()) %>%
  ungroup()
validation_rborist <- validation_rborist %>% left_join(temp, by = "userId")
rm(temp)

# add the columns for rating year and movie year
validation_rborist <- validation_rborist %>%
  mutate(rating_yr = as.numeric(format(round_date(as_datetime(timestamp), "day"), "%Y")),
         movie_yr = as.numeric(str_extract(title, "\\d{4}")))

# select the six columns that were used for training
validation_rborist <- validation_rborist %>%
  select(b_i, b_u, b_g, no.usr.ratings, rating_yr, movie_yr)

# calculate the predicted ratings for the Rborist model
validation_rborist_predict <- predict(train_rf_final, newdata = validation_rborist)

# calculate the RMSE for the Rborist model
validation_rborist_rmse <- RMSE(validation_rborist_predict, validation$rating)
```

In the code above, `validation_rborist_predict` is the vector of predicted ratings for the validation dataset using the Rborist model. The RMSE for the rborist model, stored in `validation_rborist_rmse`, is 0.87269.

The caret package provides a function, `varImp`, to evaluate variable importance for a given model. For the Rborist model this function yields the following:

```
Rborist variable importance

               Overall
b_i            100.000
b_u             66.999
no.usr.ratings   6.444
b_g              1.614
movie_yr         1.158
rating_yr        0.000
```

It is interesting to note that two variables, movie effect and user effect, far outweigh the other four in importance for predicting ratings.

Discussion

Results for both models are reasonably good. Overfitting is always a possibility but it is not likely for either of these models because neither is overly complex.

The Rborist model does not perform quite as well as the effects model but it is likely that it would fare better if trained on the entire dataset rather than the small subset of 500000 observations.

Finally, it must be acknowledged that these two models are not completely independent since the two most important variables in the Rborist model arise from the effects model.