Movielens Recommendation System

Zhao, Jiahao

5/16/2020

Abstract

For this capstone project, the author created a movie recommendation system using the MovieLens dataset (10M). The dataset included six variables, with rating as the outcome variable of analysis. The goal of this project is to predict the score a specific movie will receive for each distinct user. The data is cleaned, explored, and modelled through a ridge regression to achieve optimal RMSE

Methods and Analysis

In this section, the steps taken for data cleaning, data exploration, visualization, and the modeling approach is given and analyzed.

Data, Libraries Loading and Download

The first step is to download the dataset and load it into the R environment. Libraries are also initialized for use later. Lastly, test dataset and train dataset are partitioned. edx will be the training dataset, while validation will be the testing dataset.

Exploratory Data Analysis

The first thing is to verify the number of rows and columns in the training dataset. The code for the quizzes are given in the appendix section of this report.

```
nrow(edx) #returns number of rows
```

```
## [1] 9000055
```

```
ncol(edx) #returns number of columns
```

```
## [1] 6
```

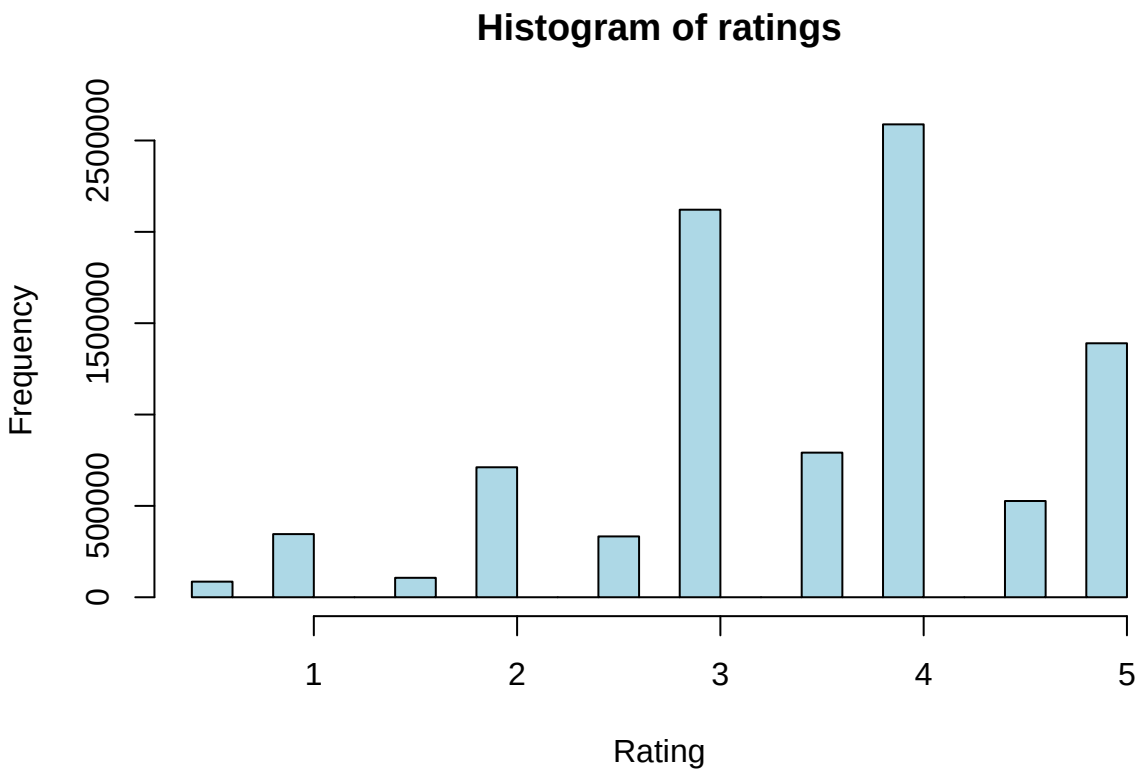
Variables

userId: the ID for the specific user movieId: the ID for the specific movie rating: the outcome variable, the rating given by the specific user timestamp: the time that the review was given title: title of the movie genres: genre of the movie

Data visualization

Next, the author creates a histogram for ratings, the outcome variable.

```
#creates histogram of ratings
hist(edx$rating, main = "Histogram of ratings", xlab = "Rating", col ="Lightblue")
```



The histogram shows that 4 and 3 are the most given scores. Scores are skewed towards higher ratings.

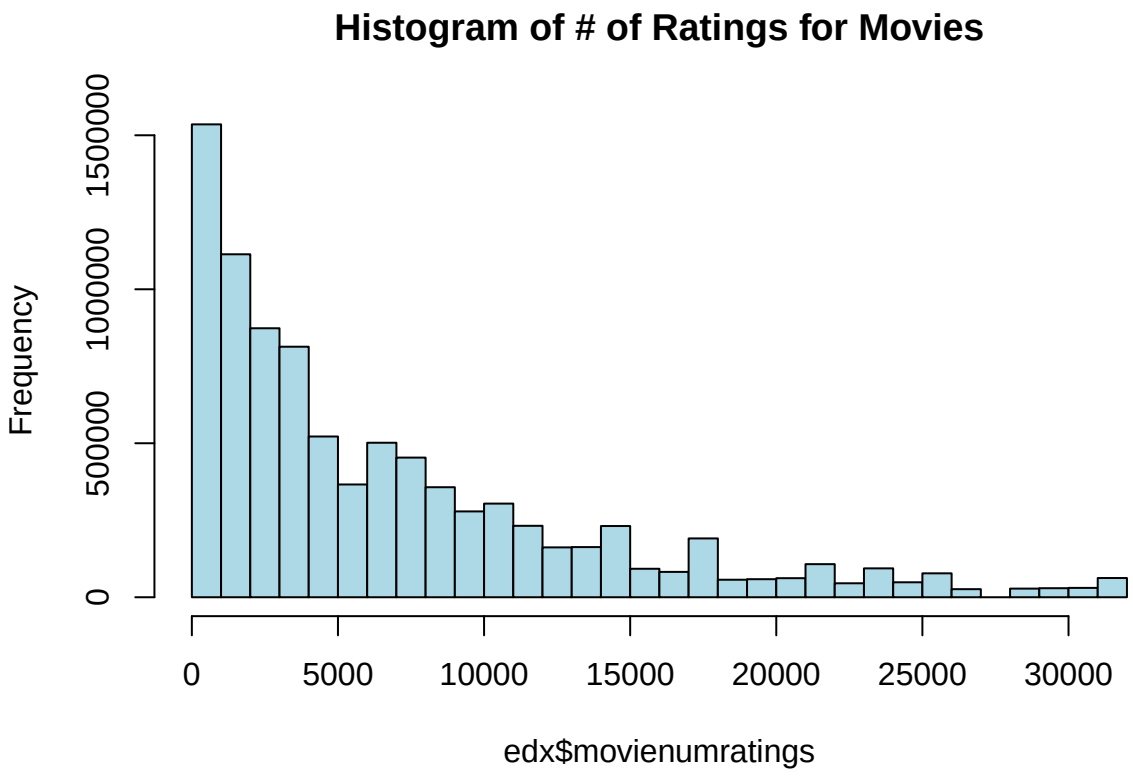
Data manipulation

With 10M records, there must be a large number of distinct movies and users (those who provided the rating). Therefore, let's analyze the average number of ratings for movies, average user number of ratings. We will also calculate movie-specific average rating and user-specific average rating.

```
edx <- edx %>% group_by(userId) %>% mutate(usernumratings = n())
# creates the column that displays the user's total number of ratings given

edx <- edx %>% group_by(movieId) %>% mutate(movienumratings = n())
# creates the column that displays the movie's total number of ratings

edx <- edx %>% group_by(movieId) %>% mutate(averagemovierating = (sum(rating))/movienumratings)
# creates the column that displays the movie's average ratings

edx <- edx %>% group_by(userId) %>% mutate(averageuserrating = (sum(rating))/usernumratings)
# creates the column that displays the user's average ratings given
```

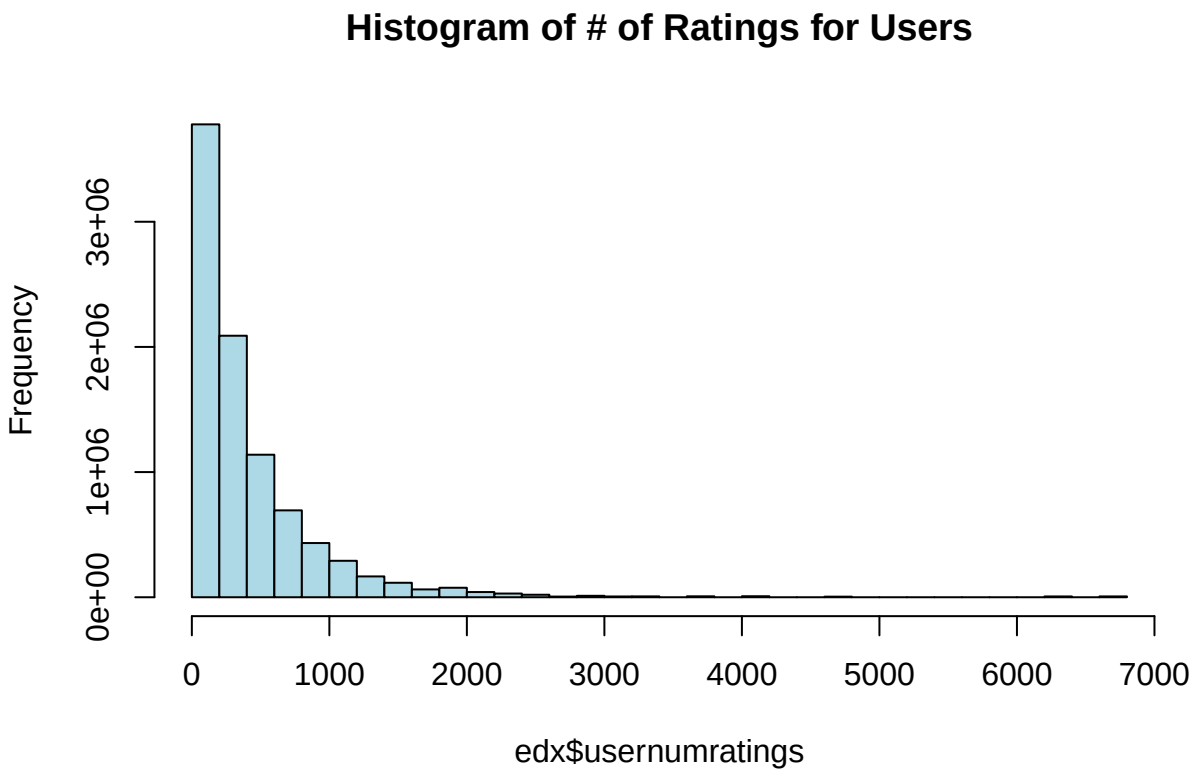
Let's see if those newly created variables follows a normal distribution. If not, they will have to be normalized.

```
hist(edx$movienumratings, main = "Histogram of # of Ratings for Movies",col="lightblue")
```




```
# creates histogram for number of ratings for movies
```

```
hist(edx$usernumratings, main = "Histogram of # of Ratings for Users", col="lightblue")
```




```
# creates histogram for number of ratings for users
```

As you can see, these variables are not at all normally distributed. We will normalize these variables for to prevent OLS assumption violation and to fit the data better.

```
mu <- mean(edx$usernumratings) # calcualtes mean
sdv <- sd(edx$usernumratings) # calculates standard deviation
edx <- edx %>% mutate(zusernumratings = (usernumratings-mu)/sdv)
# normalized usernumratings using the formula (xi-mu)/sd

mu <- mean(edx$movienumratings) # calcualtes mean
sdv <- sd(edx$movienumratings) # calculates standard deviation
edx <- edx %>% mutate(zmovienumratings = (movienumratings-mu)/sdv)
rm(mu,sdv)
# normalized movienumratings using the formula (xi-mu)/sd
```

Preparing the data for ridge regression

Multiple analysis for variables have been considered. The author performed the following procedures with the respective RMSE scores: randomForest: 2.2 OLS with one variable: 1.1 OLS with genre as dummies and interactive terms: 0.871

The OLS linear regression model performs poorly in a situation where one has a large dataset with more variables than samples. Although we do have 10M samples and only a handful of variables: Each distinct user and movie can be considered as a dummy variable. Numerical scale does not work. Thus, the OLS will not perform well in this model, especially in terms of new test data and will likely cause overtraining due to its goal of minimizing least-squares error. The best approach is the ridge regression approach. Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity as described

earlier. It is also shorter in computational time from state-of-the-art method known as the alternating least squares approach.

The glmnet package provides the functionality for ridge regression via glmnet(). Rather than accepting a formula and data frame, it requires a vector output vector and matrix of predictors.

First, we create the necessary vector and matrices.

```
y <- edx$rating # creates outcome vector
x <- edx %>% select(zusernumratings,averageuserrating,zmovienumratings,averagemovierating) %>% data.mat
```

```
## Adding missing grouping variables: `userId`
```

The cv.fit function allows us to find the best penalty rate, lambda.

```
cv_fit <- cv.glmnet(x, y, alpha = 0, lambda = (c(0.465,0.475,0.047)))
# finds the best lambda penalty rate for x input, y output, and alpha of 0 to indicate a ridge regressi
```

It is the lowest point in the curve. We can extract this values as:

```
cv_fit$lambda.min # displays the best lambda penalty rate
```

```
## [1] 0.047
```

With the lambda, we can now fit out model.

```
model <- glmnet(x, y, alpha = 0, lambda = cv_fit$lambda.min)
#fits the model of x input matrix, y output vector, alpha of 0 to indicate ridge regression, and using
```

We also prepare the testing dataset for RMSE checking.

```
validation <- validation %>% group_by(userId) %>% mutate(usernumratings = n())
# creates usernumrating column

validation <- validation %>% group_by(movieId) %>% mutate(movienumratings = n())
# creates movienumratings column
validation <- validation %>% group_by(movieId) %>% mutate(averagemovierating = (sum(rating))/movienumra
# creates average movie rating column
validation <- validation %>% group_by(userId) %>% mutate(averageuserrating = (sum(rating))/usernumratin
# calcualtes and creates average rating given by the user

mu <- mean(validation$usernumratings)
sdv <- sd(validation$usernumratings)
validation <- validation %>% mutate(zusernumratings = (usernumratings-mu)/sdv)
# normalizes usernumratings

mu <- mean(validation$movienumratings)
sdv <- sd(validation$movienumratings)
validation <- validation %>% mutate(zmovienumratings = (movienumratings-mu)/sdv)
#normalizes movienumratings

test <- validation %>% select(zusernumratings,averageuserrating,zmovienumratings,averagemovierating) %>%
```

```
## Adding missing grouping variables: `userId`
```

```
# creates matrix for input x to test model using the validation dataset.
```

Results

After comparing between RandomForest, OLS, and ridge regression, the ridge regression provides the lowest RMSE on the testing dataset of 0.8443654. The ridge regression typically results in a higher RMSE on the training dataset, but is better on new data.

```
predictions <- model %>% predict(test) %>% as.vector()
# predicts the test matrix using our model and outputs a vector.

data.frame(
     RMSE = RMSE(predictions, validation$rating),
     Rsquare = R2(predictions, validation$rating))
```

```
##        RMSE   Rsquare
## 1 0.8443566 0.3669503
```

```
# Calculates the RMSE and R-squared value using predictions and rating in the validation dataset.

RMSE = RMSE(predictions, validation$rating)
# reports RMSE
RMSE
```

```
## [1] 0.8443566
```

Conclusion

The RMSE is 0.844. Below the 0.865 requirement. I learned a great deal about machine learning through this project. I also spent a lot of time creating many instances of different algorithms before finding the one that works. The best advice for myself is to know the strength and weaknesses of each algorithms before spending hours of time running them.