DATA SCIENCE CAPSTONE PROJECT

HEMANTH KUMAR

2024-07-05

Contents

Introduction

Recommendation system provides suggestions to the users through a filtering process that is based on user preferences and browsing history. The information about the user is taken as an input. The information is taken from the input that is in the form of browsing data. This recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile. Recommender systems are beneficial to both service providers and users.

Recommendation systems use ratings that the users give an item on buying and/or using them to make specific recommendations.Companies like Amazon collect massive datasets of user ratings on the products sold to them. The datasets are subsequently used to predict a high rated items for a given user and are then recommended to the user.

Ex:- Netflix use movie ratings provided by users to predict a high rated movie for a given user and then recommended it to the user. Usually the movie ratings are on a 1-5 scale, where 5 represents an excellent movie and 1 suggests it to be a poor one.

In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. In September 2009, the

winners were announced.You can read a good summary of how the winning algorithm was put together here: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/ and a more detailed explanation here: http://www.netflixprize.com/assets/GrandPrize2009_ BPC_BellKor.pdf.

Project Information

In this project, a movie recommendation system is created using the MovieLens dataset. The version of `Movielens` included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. The entire latest `MovieLens` dataset can be found https://grouplens.org/datasets/movielens/latest/. Recommendation system is created using all the tools learnt throughout the courses in this series. `MovieLens` data is downloaded using the available code to generate the datasets.

First, the datasets will be used to answer a short quiz on the `MovieLens` data. This will give the researcher an opportunity to familiarize with the data in order to prepare for the project submission. Second, the dataset will then be used to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

The value used to evaluate algorithm performance is the `Root Mean Square Error`, or `RMSE`. RMSE is one of the most used measure of the differences between values predicted by a model and the values observed. `RMSE` is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower `RMSE` is better than a higher one. The effect of each error on `RMSE` is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on `RMSE`. Consequently, RMSE is sensitive to outliers.

A comparison of the models will be based on better accuracy. The evaluation criteria for these algorithms is a `RMSE` expected to be lower than `0.8649`.

Data Exploration : Quiz

We using MovieLens dataset for the creating a recommender system project.

The version of movielens dataset used for this final assignment contains approximately 10 Milions of movies ratings, divided in 9 Milions for training and one Milion for validation. It is a small subset of a much larger (and famous) dataset with several millions of ratings. Into the training dataset there are approximately **70,000 users** and **11,000 different movies** divided in 20 genres such as Action, Adventure, Horror, Drama, Thriller and more.

- We can download the small version of the `ml-latest.zip` file from https://grouplens.org/datasets/movielens/latest/, and unzip the file . if you directly work with dataframes

- A code to download and create `edx` and `validation` set is readily available which is to be used in this project.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################
# Note: this process could take a couple of minutes

# installing nessesary packages in our R environment
if(!require(tidyverse)) install.packages("tidyverse", repos =
"http://cran.us.rproject.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.rproject.org")

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

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

#Downloading the  movielens(ml-10m.zip) file if not avaliable in
our device
options(timeout = 120)
dl <- "ml-10M100K.zip"
if(!file.exists(dl))

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

# Unzipping the ml-10m.zip file and attaching the ratings.dat
file to  ratings_file
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

# Already we have Unzipped the ml-10m.zip file and Now simply
attaching  the movies.dat file to  movies_file
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

# Converting the ratings_files to the Dataset
ratings <- as.data.frame(str_split(read_lines(ratings_file),
fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
```

```
# Adding the coloumn names to the  rating dataset
colnames(ratings) <- c("userId", "movieId", "rating",
"timestamp")

# Coverting the coloumns values into the respective datatype
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# Converting the movies_files to the Dataset
movies <- as.data.frame(str_split(read_lines(movies_file),
fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)

# Adding the coloumn names to the  rating dataset
colnames(movies) <- c("movieId", "title", "genres")

# Coverting the coloumns values into the respective datatype
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

#Left joining between the ratings dataset with movies dataset by
the movieId
movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times =
1, p = 0.1, list = FALSE)

# Deleting the some Rows in the movielens data and defined with
the edx dataset
edx <- movielens[-test_index,]

# Extracting the Dataset with inverse of the edx dataset from
movielens data
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are
also in edx set

validation <- 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)

#Adding the new Rows to the edx from removed dataset
edx <- rbind(edx, removed)

# Removing the  extra following  dl, ratings, movies, test_index,
temp, movielens, removed data files and datasets from system
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The quiz is followed on MovieLens Dataset

1. How many rows and columns are there in the edx?

```
dim(edx)
```

2. How many zeros were given as ratings in the edx dataset?

```
edx %>% filter(rating == 0) %>% tally()
```

How many threes were given as ratings in the edx dataset?

```
edx %>% filter(rating == 3) %>% tally()
```

3. How many different movies are in the edx dataset?

```
n_distinct(edx$movieId)
```

4. How many different users are in the edx dataset?

```
n_distinct(edx$userId)
```

5. How many movie ratings are in each of the following genres in the edx dataset?

```
genres = c("Drama", "Comedy", "Thriller", "Romance")
sapply(genres, function(g) {
    sum(str_detect(edx$genres, g))
})
```

6. Which movie has the greatest number of ratings?

```
edx %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count))%>%head(5)
```

7. What are the five most given ratings in order from most to least?

```
edx %>% group_by(rating) %>% summarize(count = n()) %>%
    arrange(desc(count))%>%head(5)
```

8. True or False: In general, half star ratings are less common than whole star ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.).

```
edx %>% group_by(rating)%>%
    summarize(count=n())%>%
    arrange(desc(count))%>%
    head(10)
```

Data preprocessesing and Data visualization

Installing the nessasary packages

```
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(kableExtra)) install.packages("kableExtra")
if(!require(tidyr)) install.packages("tidyr")
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(stringr)) install.packages("stringr")
if(!require(forcats)) install.packages("forcats")
if(!require(ggplot2)) install.packages("ggplot2")

library(dplyr)
library(tidyverse)
library(kableExtra)
library(tidyr)
library(stringr)
library(forcats)
library(ggplot2)
```

- `library()`: This is a function in R that loads a package (a collection of R functions) into the current R session. The packages loaded are:
  - `tidyverse`: A collection of R packages for data science, including `dplyr`, `tidyr`, and `ggplot2`.
  - `kableExtra`: A package for creating HTML tables.
  - `tidyr`: A package for data manipulation and cleaning.
  - `stringr`: A package for working with strings.
  - `forcats`: A package for categorical data manipulation.
  - `ggplot2`: A package for data visualization.

Exploring the edx dataset details

```
summary(edx)
```

checking for any NULL values in the dataset

```
anyNA(edx)
```

counting the no.of unique users

```
edx %>% summarize(users_count = n_distinct(userId))
```

Counting the no.of unique movies

```
edx %>% summarize(users_count = n_distinct(movieId))
```

Exploring the Rating Distribution

Overall Rating distribution

```
edx %>%
  ggplot(aes(rating))+
  geom_histogram(bins= 25 ,color="lightblue")+
                   labs(title="Distribution of movie Ratings",
                        x= "Rating",
                        y="Frequency")
```

#Top 10 Movies with highest no.of ratings per movie

```
edx %>%
  group_by(title) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  head(n=10) %>%
  ggplot(aes(title, count)) +
  theme_classic()  +
  geom_col() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 7)) +
  labs(title = "Ratings Frequency Distribution Per Title - TOP 10
Movies",
       x = "Title",
       y = "Frequency")
```

Average rating for movie

```
avg_rating<-edx %>%
  group_by(title) %>%
  summarise(average_rating = mean(rating))

avg_rating %>%
  ggplot(aes(average_rating)) +
  theme_classic()  +
  geom_histogram(bins=12) +
  labs(title = "Average rating for movieId",
       x = "Mean",
       y = "Frequency")
```

#top 10 movies with highest rating

```
top_movies<- avg_rating %>%
  arrange(desc(average_rating)) %>%
```

```
  head(n=10)
top_movies
```

#meadian distribution of rating with title

```
edx %>%
  group_by(title) %>%
  summarise(median = median(rating)) %>%
  ggplot(aes(median)) +
  theme_classic()  +
  geom_histogram(bins=12) +
  labs(title = "Median Distribution per Title",
       x = "Median",
       y = "Frequency")
```

Model Training

Defining the RSME

```
# The RMSE function that will be used in this project is:
RMSE <- function(true_ratings = NULL, predicted_ratings = NULL) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

- `rmse()`: This is the function being defined. It calculates the Root Mean Squared Error (RMSE) between two vectors of numbers.
  - `true_ratings`: The true ratings (actual values).
  - `predicted_ratings`: The predicted ratings (predicted values).
  - `sqrt()`: The square root function, used to calculate the square root of a number.
  - `mean()`: The mean function, used to calculate the average of a vector of numbers.
  - `(true_ratings - predicted_ratings)^2`: The difference between the true and predicted ratings, squared.

Naive-based model

Calculating the average rating of all the movies using train set. We can see that if we predict all unknown ratings with μ we obtain the following RMSE.

```
{r Naive-based model, echo=FALSE} # Compute the dataset's mean rating
mu <- mean(edx$rating) mu
```

We observe the first model indicates that movie rating is `>3.5`.

Movie-based model

We know from experience that some movies are just generally rated higher than others.Lets see if there will an improvement by adding the term b_i to represent the average rating for movie i.

```
{r movie-based model, echo=TRUE} # Simple model taking into account the movie effect b_i # Subtract the rating minus the mean for each rating for a movie # Plot no.of movies with the computed b_i
movie_avgs <- edx %>%   group_by(movieId) %>%   summarize(b_i = mean(rating - mu)) movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"), ylab = "No.of movies", main = "No.of movies with the computed b_i")
```

Movie + User-effect model

```{r movie-user-based model, echo=TRUE} ## Movie and user effect model ##

# Plot penaly term user effect

user_avgs<- edx %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>% filter(n() >= 100) %>% summarize(b_u = mean(rating - mu - b_i)) user_avgs%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))

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

## Movie + Title + User-effect model

```{r movie-title-user-based model, echo=TRUE}
#just the average of ratings
mu <- mean(edx$rating)
# Calculate the average by title
title_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(title) %>%
  summarize(b_tt = mean(rating - mu - b_i - b_u))
#compute the predicted ratings on test set
predicted_ratings_movie_title_user_avg <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(title_avgs, by='title') %>%
  mutate(pred = mu + b_i + b_u + b_tt) %>%
  pull(pred)
```
```

Movie + User + Regularization

To better our models above and ensure we are converging to the best solution let us add a regularization constant for our movie rating and user-specific model. The purpose of this is to penalize large estimates that come from small sample sizes. Therefore, our estimates will try its best to guess the correct rating while being punished if the movie rating, user-specific is too large.

This method is a little more complicated than the prior two methods. That is we want to also validate how much we want to regularize the movie-rating and user-specific effects. We will try several regularization models with our regularization constant `(lambda)` at different values. We will define the function to obtain `RMSEs` here and apply it in our results section:

```{r minimizing RMSE using appropriate lambda value, echo=TRUE}

## lambdas is a tuning parameter

## use cross-validation to choose it.

lambdas <- seq(0, 10, 0.25)

## For each lambda,calculate b_i & b_u, the rating prediction & testing

rmses <- sapply(lambdas, function(l){

mu <- mean(edx$rating)

b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n()+l))

b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>% summarize(b_u = sum(rating - b_i - mu)/(n()+l))

predicted_ratings <- test %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>% pull(pred)

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


# Model Evalution {#sec-model-evalution}

In this section we will see how well our models worked to our test set and then we validate the best model to unseen data in the `Validation` set.

## RMSE for Naive-based model
```

```{r echo=FALSE}
# Test results based on simple prediction
rmse1 <- RMSE(validation$rating, mu)
rmse1

# Check results
# Save prediction in data frame
rmse_results <- data.frame(method = "Average movie rating model", RMSE
=rmse1)
rmse_results %>% knitr::kable()
```

RMSE for Movie effect model

{r echo=FALSE} # Test and save rmse results predicted_ratings <- mu + validation %>% left_join(movie_avgs, by='movieId') %>% pull(b_i) rmse2 <- RMSE(predicted_ratings, validation$rating) rmse_results <- bind_rows(rmse_results,data_frame(method="Movie effect model", RMSE =rmse2)) # Check results rmse_results %>% knitr::kable()

Based on the RMSE result obtained, `0.9439087` indicates an improvement. But can we make it much better than this?

We can see that these estimates vary as indicated on the plot below:

{r qplot for movie averages, echo=FALSE} qplot(b_i, geom = "histogram", color = I("black"), fill=I("brown"), bins=40, data = movie_avgs)

RMSE for Movie + user-specific model

```{r echo=FALSE} #calculate RMSE} # Test and save the rmse results predicted_ratings <- validation%>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs, by='userId') %>% mutate(pred = mu + b_i + b_u) %>% pull(pred)

rmse3 <- RMSE(predicted_ratings, validation$rating) rmse_results <- bind_rows(rmse_results,data_frame(method="Movie + user effect model",RMSE =rmse3))

## Check result

rmse_results %>% knitr::kable()

This seems to be a better model with an RMSE value of `0.8653488`.

We can see that these estimates vary as indicated on the plot below:

```{r qplot for user avaerages, echo=FALSE}
qplot(b_u, geom = "histogram", color = I("black"), fill=I("brown"), bins=40, data = user_avgs)
```
```

RMSE for Movie + user-specific + regularization model

This result is a little more complicated than the prior two methods. That is we want to also validate how much we want to regularize the movie-rating and user-specific effects. We will try several regularization models with our regularization constant (lambda) at different values:

```{r echo=FALSE} # Plot lambdas vs rmses to select the optimal lambdas
qplot(lambdas, rmses)

## The optimal lambda

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


Our minimum error is found with a lambda value of 5.25:

```{r echo=FALSE}
# Test and save results

rmse_results <- bind_rows(rmse_results,data_frame(method="Regularised
+ movie + user effect model",RMSE = min(rmses)))

# Check result
rmse_results %>% knitr::kable()
```

All Models Results

The table below summarizes our RMSE values given our models explored:

```
{r echo=FALSE, warning=FALSE} #### Results ####
# RMSE results overview
rmse_results %>% knitr::kable()
```

As we can see the regularization model performed best when lambda was set to `5.25` with an RMSE of `0.8571`. Let's proceed to perform model validation based on this model.

Conclusion

After training different models, it's very clear that `movieId` and `userId` contribute more in the predictor. Without regularization, the model can archieves and overtakes the desidered peformance, but the good and applying best regularization it make possible to reach a RSME of **`0.8571`** defined that is the best result
```

References

All material in this project is credited to Professor Rafael Irizarry and his team at HarvardX's Data Science course. Most material was learned through his course, book, and github:

1.Irizarry,R 2021 ***Introduction to Data Science: Data Analysis and Prediction Algorithms with R*** ,github page,accessed 22 January 2021, https://rafalab.github.io/dsbook/

Another great resource was prior movie recommendation projects. Specifically, one project was referenced:

2. The work done by Eddwin Cheteni (I try data exploration techniques such as similarity measures)