Report on Movilens

Ruben Rodriguez de la Fuente

February 10, 2022

Introduction

The movieLens data set includes 10 million movie ratings, resulting from 72,000 users rating 10,000 movies. Ratings are on a 1 to 5 scale. Our goal is to use this data to create a recommendation model that would predict the rating a user would give a movie based on different features, namely average rating for all movies, average rating by movie, by user or genre. The model will be fine-tuned using regularization, to reduce the noise introduced my movies with small sample sizes. The metric to evaluate the model will be root mean square error (RMSE), which measures the variance of prediction errors.

Methods and analysis

First we prepare a function to calculate our metric of interest. We then load the libraries we need, prepare the data set and split into training, test and validation test. We will begin with a naive model always predicting the average rating use as baseline. We will add features one by one and see whether they improve performance. Finally, we will use regularization to reduce the effect of noise

Validation metric

```
# Prepare function to calculate root mean square error
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))}
```

Libraries for analysis

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

Data set preparation

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

dl <- tempfile()
download.file("http://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")

# if using R 4.0 or later:
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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
```

```
  semi_join(edx, by = "userId")

#create a new partition for train and test
set.seed(755)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- movielens[-test_index,]
test_set <- movielens[test_index,]

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Feature extraction and modeling

Here we extract features and check the importance one by one. First we start by simply using the general rating average to get a baseline. The RMSE is 1.06

```
mu <- mean(train_set$rating) #general rating average
RMSE(test_set$rating, mu)
```

```
## [1] 1.060047
```

Next we add the average rating by movie to our model. RSME decreases to 0.94

```
movie_avgs <- train_set %>% #average rating by movie
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.9441338
```

Now we add the average rating by user to our model. RSME decreases to 0.86

```
user_avgs <- train_set %>% #average rating by users
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.8652017
```

Next we try adding average rating by genre to our model. RSME does not improve. This could be due to the fact that the genre labels are a bit noisy, with multiple genres (action, comedy, romance) attached to the same movie. It could be interesting to try some cleaning, like consolidating labels. This feature is discarded for the model.

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i))

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

```
## [1] 0.8652017
```

Now we will try regularization on our best model so far (movie/user averages). We will apply a penalty (lambda) to movies with small sample sizes. We find the optimal value of lambda by trying several values, from 0 to 10 in 0.25 increments. We find that our best lambda is 4.75

```
#Regularization
lambdas <- seq(0, 10, 0.25)#select optimal lambda
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

lambda <- lambdas[which.min(rmses)] #pick lambda that minimizes RMSE

mu <- mean(train_set$rating)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
print ('final model on test set')
```

```
## [1] "final model on test set"
```

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

```
## [1] 0.8646509
```

Results

The table below shows the comparisons between different models. The one using movie and user averages with regularization is the best one.

```
models<-c('naive', 'movie avg', 'movie+user avg', 'movie+user+genre avg', 'movie+user avg regularized')
scores<-c(1.06, 0.944, 0.865, 0.865, 0.864)
df<-data.frame(models, scores)
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.1.2
```

```
kable(df, caption='model comparison')
```

Table 1: model comparison

models	scores
naive	1.060
movie avg	0.944
movie+user avg	0.865
movie+user+genre avg	0.865
movie+user avg regularized	0.864

We try our best model on the validation set and get an RMSE of 0.858.

```
predicted_ratings <- validation %>% #try on validation test set
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
print ('final model on validation set')
```

```
## [1] "final model on validation set"
```

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

```
## [1] 0.8582879
```

Conclusions and Future work

We were able to improve the baseline model by adding average rating by movie and user and using regularization. The model could be further improved with genre labels consolidation. Another interesting idea would be merging with other data sets, like the IMDB ones (https://www.imdb.com/interfaces/), so that we can also add features as director, screenplay writer or lead actor.