edxCapstone

Jose Arizpe

29 December 2019

Movielens project / edx Data Science capstone project

The following code follows the methodology used in the course for fitting the movielens data, based on producing predictions based on the following formula:

- $Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$

where

- $Y_{u,i}$ - is the prediction for movie i and user u
- μ - is the mean of ratings
- b_i - is the movie effect
- b_u - is the user effect
- $\varepsilon_{u,i}$ - is the residual error after taking into account the effects above

The method above has been enhanced with the following additions:

1- Introduction of regularisation based on penalised least squares

2- Break-up of the movielens "genres" field into columns, one column per genre containing logical values, so this information will also be available to the fitting process

3- Fitting of the residuals using the augmented file (including the expanded genres) using the "rpart" tree-based model

The highest improvement on RMSE is obtained when $\varepsilon_{u,i}$, the residual error, is estimated on its own and apart from the other variables in the formula, using for this purpose the rpart tree-based model. The fitting of residuals is in turn greatly enhanced by the splitting of the "genres" field into individual logical columns for each genre encountered in the data.

The application of regularisation based on penalised least squares also improves RMSE but in a much more modest way than the fitting of residuals.

`The resulting RMSE (0.426187) can be seen at the bottom of the report.`

Here are the steps followed to obtain this result:

Load and prepare data

1- Load data using the code provided

Create edx and validation sets . . . . . .

2- Extract the year from the movie title

```
# CREATE DATE FIELD
edx <- edx %>% mutate(date = as_datetime(timestamp))
# EXTRACT YEAR AND TITLE FROM ORIGINAL "title" COLUMN
#    Extract the year into vector "y"
y <- str_match(edx$title, "^(.*)\\s\\((\\d{4})\\)$")[,3]
```

```
edx$year <- y ## Attach "y" as a new column to "edx"
#    Eliminate year data from title
movieTitles <- str_match(edx$title, "^(.*)\\s\\((\\d{4})\\)$")[,2]
edx$title <- movieTitles ## Replace title with the stripped down new title
```

3- Break-up the "Genres" column

Into as many columns as there are defined genres with logical values for each movie highligthing the genres which those movies have been classsified against.

```
# FIND OUT ALL DIFFERENT GENRES THAT EXIST IN THE LIST OF MOVIES AND PUT THEM IN A
# TABLE CALLED allGenres
#   Create new data frame including only "movieId" and "genres"
movieGenres <- edx %>% select(movieId, genres) %>% unique()
movieGenres <- suppressWarnings(movieGenres %>% separate("genres", c("g1", "g2", "g3",
                                                                     "g4", "g5", "g6",
                                                                     "g7", "g8", "g9",
                                                                     "g10"), "\\|"))
#   separate genres into different columns, assume no more than 10 genres per movie.
#   Create 1-n table for "movieId" and "genre". Use intermediate table t
t <- gather(movieGenres, "key", "genre", -movieId, na.rm = TRUE)
allGenres <- t %>% select(genre) %>% unique() ## List all different genres found
#   Delete records with text in parenthesis, assuming these are not valid genre names
allGenres <- allGenres %>% filter(!str_detect(allGenres$genre, "^\\(.*\\)$"))
#   Create colums with genres
for (g in allGenres[,1]) {
  edx <- cbind(edx, data.frame(nom = str_detect(edx$genres, g)))
  names(edx)[names(edx) == "nom"] <- str_replace(tolower(g), "-", "_")
}
```

(Note that this processing makes the assumption that no movie is identified with more than 10 genres. Also it assumes that genres in parentheses are not genre names but rather it means no genre has been identified.)

Here is a list of all genres found

```
allGenres
```

```
##          genre
## 1       Comedy
## 2       Action
## 3     Children
## 4    Adventure
## 5    Animation
## 6        Drama
## 7        Crime
## 8       Sci-Fi
## 9       Horror
## 10    Thriller
## 11   Film-Noir
## 12     Mystery
## 13     Western
## 14 Documentary
## 15     Romance
## 16     Fantasy
## 17     Musical
## 18         War
## 19        IMAX
```

And here is an extract of the extended "edx" data including the added columns

```
as_tibble(head(edx))
```

```
## # A tibble: 6 x 27
##   userId movieId rating timestamp title genres date                year
##    <int>   <dbl>  <dbl>     <int> <chr> <chr>  <dttm>              <chr>
## 1      1     122      5 838985046 Boom~ Comed~ 1996-08-02 11:24:06 1992
## 2      1     185      5 838983525 Net,~ Actio~ 1996-08-02 10:58:45 1995
## 3      1     292      5 838983421 Outb~ Actio~ 1996-08-02 10:57:01 1995
## 4      1     316      5 838983392 Star~ Actio~ 1996-08-02 10:56:32 1994
## 5      1     329      5 838983392 Star~ Actio~ 1996-08-02 10:56:32 1994
## 6      1     355      5 838984474 Flin~ Child~ 1996-08-02 11:14:34 1994
## # ... with 19 more variables: comedy <lgl>, action <lgl>, children <lgl>,
## #   adventure <lgl>, animation <lgl>, drama <lgl>, crime <lgl>,
## #   sci_fi <lgl>, horror <lgl>, thriller <lgl>, film_noir <lgl>,
## #   mystery <lgl>, western <lgl>, documentary <lgl>, romance <lgl>,
## #   fantasy <lgl>, musical <lgl>, war <lgl>, imax <lgl>
```

4- Repeat the previous steps for the "validation" set

. . . . . .

Create the model / start fitting

5- Naive approach

Start with naive approach as initial benchmark to measure progress against:

```
# Naïve RSME
mu_hat <- mean(edx$rating)
naive_rmse <- RMSE(validation$rating, mu_hat)
```

```
## RMSE naive method = 1.06120181029262
```

6- Try rpart as overarching method for predicting ratings

Predict ratings using a tree-based method due to the number of columns that have been generated. I have tried various methods but all of them end up running out of memory except "rpart". Here is the first shot at that:**

```
set.seed(1)
library(rpart) # Fit ratings using "rpart"
fit <- rpart(rating~., data = edx)
predicted_ratings <- predict(fit, validation)
model_0_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
## RMSE fit ratings using rpart = 0.957880109409465
```

As can be seen, initial results with rpart for directly predicting the end-ratings are not great and do not achieve many points, so go back to the method used in the course, using the following formula as explained above:

$(Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i})$

7- Add movie effects (b_i) to naive estimate (μ):

```
mu <- mean(edx$rating)
movie_avgs <- edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
predicted_ratings <- mu + validation %>% left_join(movie_avgs, by='movieId') %>%
```

```
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
## RMSE from adding movie effect to naive = 0.943908662806309
```

8- Add user effects (b_u) to naive estimate (μ) + movie effects (b_i):

```
user_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>% mutate(pred = mu + b_i + b_u) %>% pull(pred)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
```

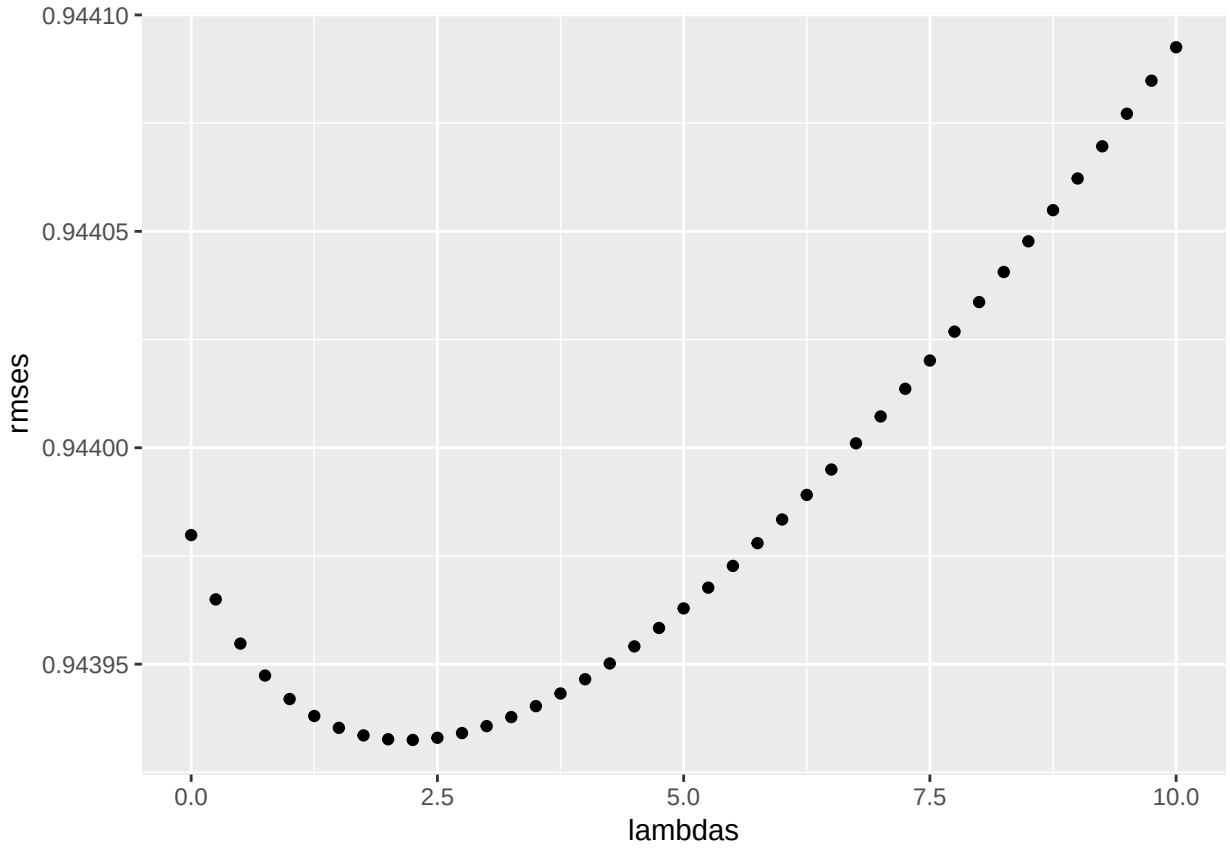
```
## RMSE from adding movie effect and user effect to naive = 0.865348824577316
```

9- Apply regularisation to the naive (μ) + movie effects (b_i) estimate

Apply regularisation based on penalised least squares, and see if this improves the results. For this the data has to be partitioned in order to fin the lambda value that yields the lowest RMSE:

```
# Movie effect with regularisation
# Partition data
set.seed(1)
test_index <- createDataPartition(edx$rating, times=1, p=0.9, list=FALSE)
edx_test <- edx[-test_index,]
edx_train <- edx[test_index,]
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

# Determine the lambda
lambdas <- seq(0, 10, 0.25)
mu <- mean(edx_train$rating)
just_the_sum <- edx_train %>% group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l1){
  predicted_ratings <- edx_test %>% left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l1)) %>% mutate(pred = mu + b_i) %>% pull(pred)
  return(RMSE(predicted_ratings, edx_test$rating))
})
qplot(lambdas, rmses)
```




```
l1 <- lambdas[which.min(rmses)]
# Calculate RMSE for better lambda
mu <- mean(edx$rating)
just_the_sum <- edx %>% group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
predicted_ratings <- validation %>%
  left_join(just_the_sum, by='movieId') %>% mutate(b_i = s/(n_i+l1)) %>%
  mutate(pred = mu + b_i) %>% pull(pred)
model_6_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
## RMSE from naive + regularised movie effect = 0.943852139888902
```

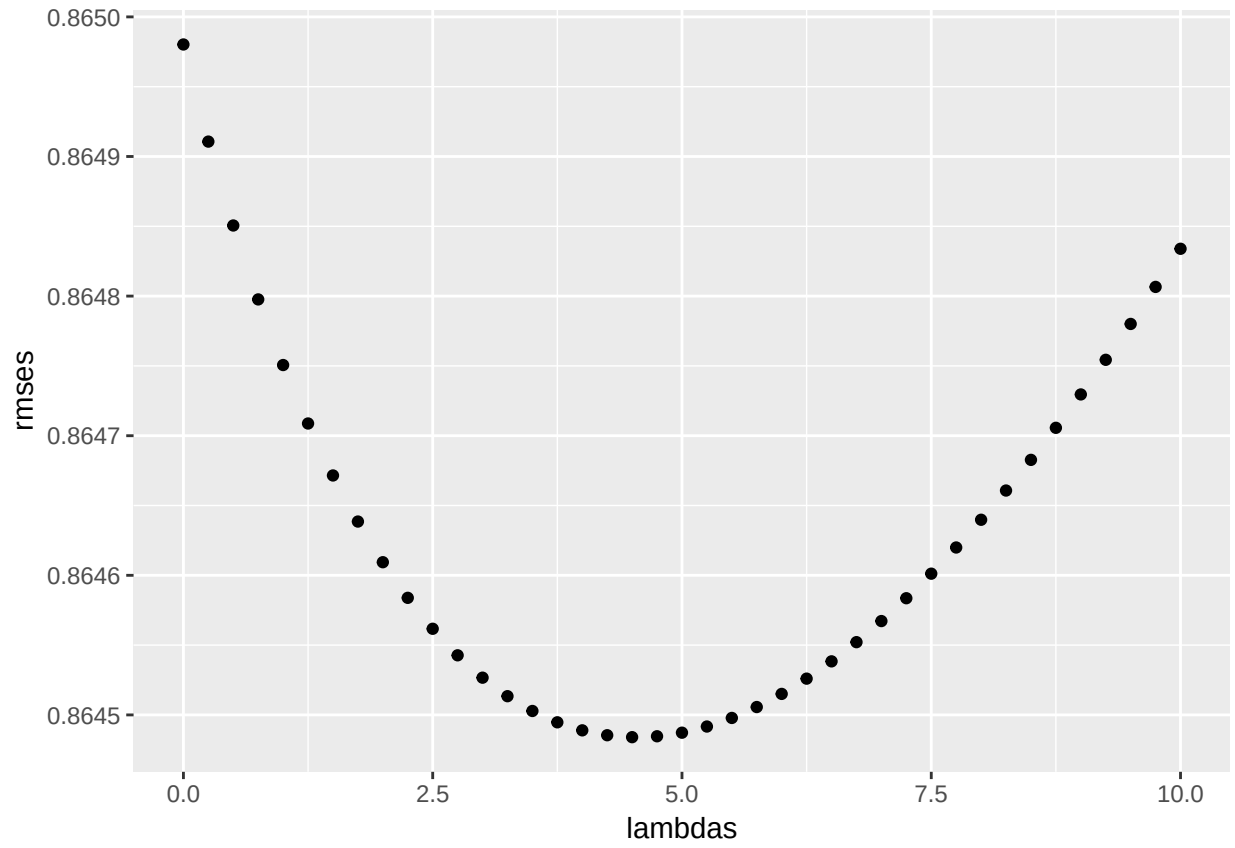
```
## (Best lambda = 2.25)
```

10- Apply regularisation to the naive (μ) + movie effects (b_i) + user effects (b_u) estimate

Apply regularisation based on penalised least squares, and see if this improves the results. For this the data has to be partitioned in order to fin the lambda value that yields the lowest RMSE:

```
# Movie and user effect with regularisation
# Partition data
set.seed(1)
test_index <- createDataPartition(edx$rating, times=1, p=0.9, list=FALSE)
edx_test <- edx[-test_index,]
edx_train <- edx[test_index,]
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

```r
# Determine the lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l2){
  mu <- mean(edx_train$rating)
  b_i <- edx_train %>% group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l2))
  b_u <- edx_train %>% left_join(b_i, by="movieId") %>%
    group_by(userId) %>% summarize(b_u = sum(rating - b_i - mu)/(n()+l2))
  predicted_ratings <- edx_test %>% left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, edx_test$rating))
})
qplot(lambdas, rmses)
```




```r
l2 <- lambdas[which.min(rmses)]
# Calculate RMSE for better lambda
mu <- mean(edx$rating)
b_i <- edx %>% group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l2))
b_u <- edx %>% left_join(b_i, by="movieId") %>%
  group_by(userId) %>% summarize(b_u = sum(rating - b_i - mu)/(n()+l2))
predicted_ratings <- validation %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

```
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
## RMSE from naive + regularised movie effect + user effect = 0.8648242289282
```

```
## (Best lambda = 4.5)
```

11- Model the residuals ($\varepsilon_{u,i}$) using rpart

Use the complete data breadth, initial columns + calculated year + additional genre columns created above.

As mentioned before, I have tried a variety of training methods but all of them run into memory problems except "rpart" which also yields an important decrease in RSME as can be seen below.

Code and RMSE results follow:

```
# NOW MODEL RESIDUALS INDEPENDENTLY ON TOP OF THE PREVIOUS MODELS USING rpart ON THE
# NEW GENRE COLUMNS CREATED BEFORE
# First identify the winning parameters for all calculations to follow
mu <- mean(edx$rating)
b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n()+12))
b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+12))
# Create residuals table with residuals on ratings made by users
residuals <- edx %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>% mutate(residual = rating - mu - b_i - b_u) %>%
  pull(residual)
edx <- bind_cols(edx, data.frame(residuals = residuals))
library(rpart) # Fit residuals using "rpart"
fit <- rpart(residuals~., data = edx)

# Calculate predicted ratings using mu + b_i + b_u + residuals, i.e. adding residuals
# to previous best estimate
r_hat <- predict(fit, validation)
validation <- bind_cols(validation, data.frame(residuals = r_hat))

predicted_ratings <- validation %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u + residuals) %>%
  pull(pred)
model_4_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
## RMSE from naive + regularised movie & user effects + rpart residuals fit = 0.426187656223384
```

12- Model the residuals ($\varepsilon_{u,i}$) without previous regularisation

Just to be sure, and taking into account that the RMSE improves very little with Regularisation, model the residuals same as in the previus step but using the non-regularised naive + movie + user effects results.

```
# NOW MODEL RESIDUALS SAME AS ABOVE BUT WITHOUT REGULARISATION IN THE PREVIOUS STEP
# First identify the winning parameters for all calculations to follow
mu <- mean(edx$rating)
b_i <- edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu))
# Create residuals table with residuals on ratings made by users
residuals <- edx %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>% mutate(residual = rating - mu - b_i - b_u) %>%
  pull(residual)
edx <- edx[ ,-which(names(edx)=="residuals")]
```

```
edx <- bind_cols(edx, data.frame(residuals = residuals))
fit <- rpart(residuals~., data = edx) # Fit residuals using "rpart"

# Calcuate predicted ratings using mu + b_i + b_u + residuals, i.e. adding residuals
# to previous best estimate
r_hat <- predict(fit, validation)
validation <- validation[ ,-which(names(validation)=="residuals")]
validation <- bind_cols(validation, data.frame(residuals = r_hat))

predicted_ratings <- validation %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u + residuals) %>%
  pull(pred)
model_5_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
## RMSE from naive + movie & user effects + rpart residuals fit = 0.436926766613497
```

Show the results

13- Results summary:

```
##                                                          method      rmse
## 1 Naive + regularised movie & user effects + rpart residuals fit 0.4261877
## 2             Naive + movie & user effects + rpart residuals fit 0.4369268
## 3                 Naive + regularised movie & user effects Model 0.8648242
## 4                             Naive + movie & User effects Model 0.8653488
## 5                       Naive + regularised movie effects Model 0.9438521
## 6                                   Naive + movie effects Model 0.9439087
## 7                                            rpart ratings fit 0.9578801
## 8                                                        Naive 1.0612018
```

```
## Final RMSE = 0.426187656223384
```

```
## delivered by 'Naive + regularised movie & user effects + rpart residuals fit'
```