PH125_9 MovieLens Capstone Project

Mario De Toma

February 28, 2019

Overview

Objectives

Motivation) This project has been conducted as part of the Data Science Professional Certification path provided by HarvardX, an online learning initiative of Harvard University through edX. In particular this is the first data science projet to submit for PH125.9x course denominated "Data Science: Capstone".

Project objective) Project objective is to build a movie recommender system using MovieLens dataset. In particular the main objective is to demonstrate the ability to predict movie ratings using machine learning technique. Despite the fact ratings scale is discrete ranging from 0.5 to 5 with a step size of half, in the study ratings will be considered as real number. The results of predictions is evaluated using RMSE, Root Mean Squared Error, metric.

Research question) The question this project is answering to is: Is it possible to predict the rating that a particular user will give to a specific movie and with which error on average?

Dataset) GroupLens research lab geneated a movie ratings database with over 20 milion ratings for 27,000 movies by more than 138,000 users. The dataset for this project is the MovieLens dataset with 10 milions row downloaded from GroupLens website [1]. This is a subset dataset containing 10,000,000 movie ratings.

Background and related works

Starting point for conducting this study is the recommender model decribed by Professor Rafael Irizarry in the PH125_8 edX course on Machine Learning and in his book Introduction to Data Science [2]. Further base for investigation was the provided link to Netflix Prize [3].

Theory) Matrix factorization is the state-of-the-art solution for sparse data problem, although it has become widely known since Netflix Prize Challenge. Matrix factorization is simply a family of mathematical operations for matrices in linear algebra. To be specific, a matrix factorization is a factorization of a matrix into a product of matrices. In the case of collaborative filtering, matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. One matrix can be seen as the user matrix where rows represent users and columns are latent factors. The other matrix is the item matrix where rows are latent factors and columns represent items. In the sparse user-item interaction matrix, the predicted rating user u will give item i is computed as:

$$r_{ui} = \sum_{f=0}^{nfactors} h_{u,f} \cdot w_{f,i}$$

Rating of item i given by user u can be expressed as a dot product of the user latent vector and the item latent vector.

Overview and outline

The study demonstrates that using matrix factorization reduces the error measured with RMSE consistently. The study will be conducted in 3 different stages by modeling the group level effects of user, movie and movie year, then regularizing the parameters of this simple model and at last adding the latent factor component.

This report is articulated in the following few sections:

- *Methods and Analysis*: where the Movielns dataset has been explored in order to find some insight, then the model has been proposed and the design of the study explained. Finally the modeling will be described in details.
- *Results*: showing actual results achieved
- *Conclusions*: summarizing achievement, discussing the project and indicating potential model improvement
- *Reproducibility*: providing information related to the reproducibility of the analysis including computation considerations, HW and SW stack used.

Methods and Analysis

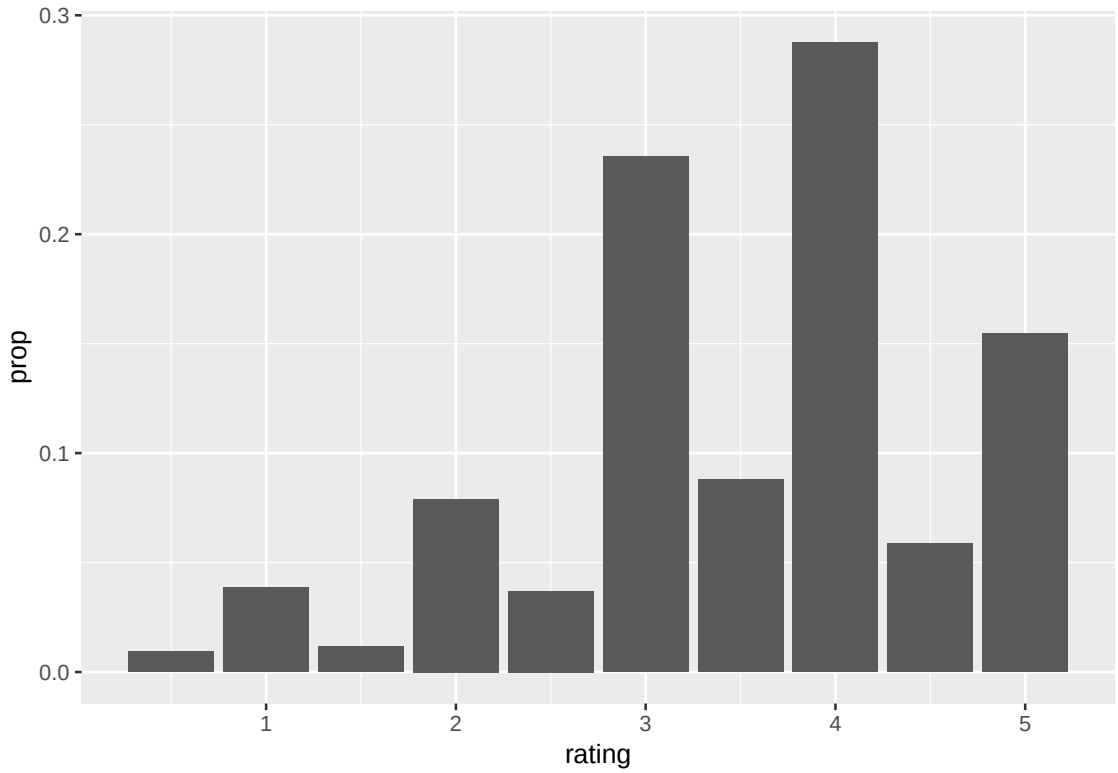
Exploratory Data Analysis

MovieLens data is loaded into R and partitioned in training set called edX and validation set called validation executing the script provided by HarvardX.

edx dataset contains rating. for 69878 users and 10677 movies.

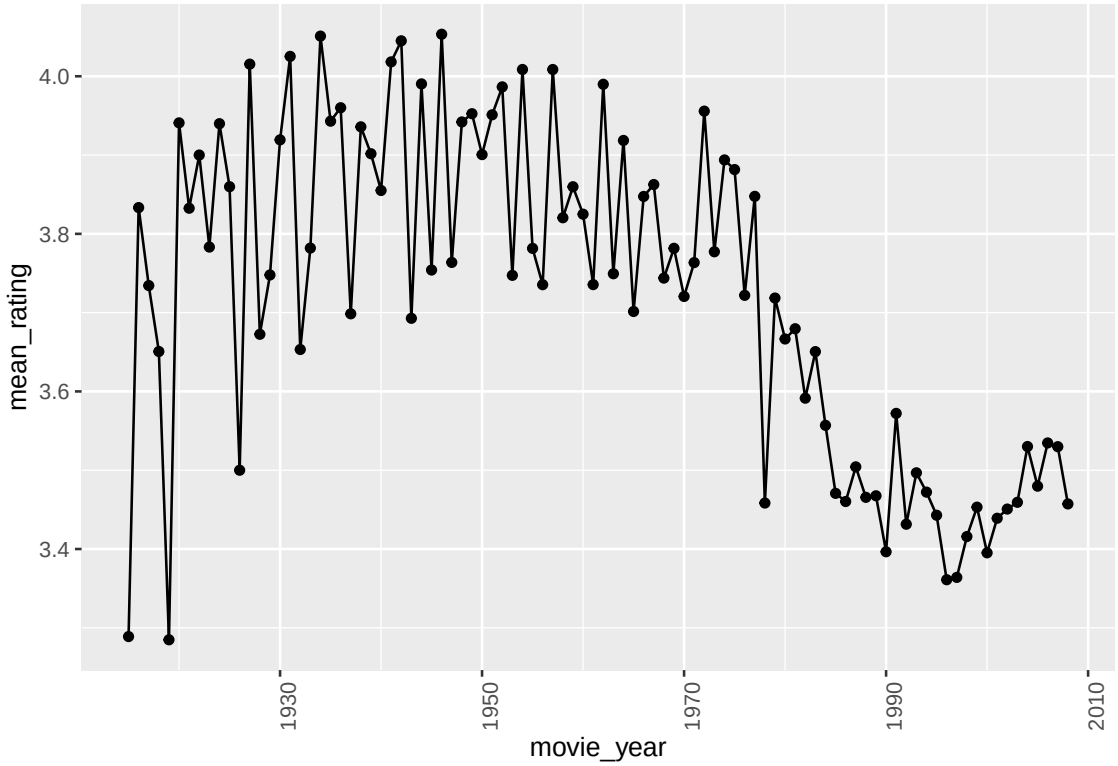
The overall ratings discrete distribution shows that half points are relativly less probable and that higher rating are most likely

```
edx %>% group_by(rating) %>% summarise(prop = n()/nrow(edx)) %>%
  ggplot(aes(rating, prop)) + geom_col()
```



According to recommender system literature we expect that user preference and movie overall quality influence the ratings. In order to investigate if any time effect could have any impact, the trend of rating mean along movie year can be explored.

```
edx %>%
  mutate(movie_year = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
  group_by(movie_year) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(mapping = aes(x = movie_year, y = mean_rating)) +
  geom_point() + geom_line() +
  theme(axis.text.x = element_text(angle = 90))
```



The plot shows some effect related to the year of the movie.

Another interesting insight about time is that no half point value has been rated before 2003. This impact the probability of finding a half point rate overall (see above).

```
library(lubridate)
edx %>% filter(rating %in% c(0.5, 1.5, 2.5, 3.5, 4.5)) %>%
  mutate(year_rated = year(as_datetime(timestamp))) %>%
  group_by(year_rated) %>%
  summarise(half_rated = n()) %>%
  arrange(half_rated)
```

```
## # A tibble: 7 x 2
##   year_rated half_rated
##        <dbl>      <int>
## 1       2009       5789
## 2       2003     179339
## 3       2007     273652
## 4       2006     301310
## 5       2008     305637
## 6       2004     307331
## 7       2005     470112
```

Proposed model

The proposed model should take into considaration the global mean μ effect, the group level effects for movie, user and movie year and will add the latent factor effect.

$$r_{u,m} = \mu + b_m + b_u + b_y + \sum_{n=1}^{nfactors} h_{u,f} \cdot w_{f,m} + \epsilon$$

with nfactors = 50.

Study design

The study design foresees to train the model in edx dataset and to assess results on the validation set. Validation set is only the 10% of the data but it counts as much as 999999 observations. The prediction results will be measured by Root Mean Sqare Error (RMSE). Recall that better the model lower the RMSE.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The model could be considered useful if the RMSE metric reach a value below the one reached with the null model based only on the overall mean 3.5124652.

```
rmse_null <- RMSE(validation$rating, mean(edx$rating))
rmse_results <- tibble(method = 'overall mean model',
                       RMSE = rmse_null)
rmse_results %>% knitr::kable()
```

method	RMSE
overall mean model	1.061202

The model will be constructed in 3 steps

- first modeling effects,
- then regularizing the model
- and finally adding the latent factor effect provided by the rating matrix factorization.

Modeling effects

First we build the simple additive model $r_{u,m} = \mu + b_m + b_u + b_y$ where

- μ is the overall mean
- b_m is the group level effect by movie
- b_u is the group level effect by user

- b_y is the group level effect by movie year (this feature is extracted from the title variable)

```
mu <- mean(edx$rating)
movie_effect <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu))
user_effect <- edx %>%
  left_join(movie_effect, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - mu - b_movie))
year_effect <- edx %>%
  mutate(year_movie = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
  left_join(movie_effect, by = 'movieId') %>%
  left_join(user_effect, by = 'userId') %>%
  group_by(year_movie) %>%
  summarise(b_year = mean(rating - mu - b_movie - b_user))

# recommender prediction
predicted_real_ratings <- validation %>%
  mutate(year_movie = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
  left_join(movie_effect, by = 'movieId') %>%
  left_join(user_effect, by = 'userId') %>%
  left_join(year_effect, by = 'year_movie') %>%
  mutate(predicted_real = mu + b_movie + b_user + b_year) %>%
  .$predicted_real


# evaluating root mean squared error
rmse_0 <- RMSE(true_ratings = validation$rating, predicted_ratings = predicted_real_ratings)
rmse_results <- bind_rows(rmse_results,
                          tibble(method = 'movie + user + movie_year effects',
                                 RMSE = rmse_0))

# remove objects to free memory
rm(movie_effect, user_effect, year_effect, rmse_0)
gc()
```

Regularizing effect model

Minimizing ridge regression objective function, it is demonstrated the every effect parameter can be calculated (eg for movie effect) as

$$\hat{b_m}(\lambda) = \frac{1}{\lambda + n_m} \sum_{u}^{n_m} (Y_{u,m} - \hat{\mu})$$

where λ is a tuning parameter to be found through validation.

```
# search for best lambda
lambdas <- seq(0,10, 0.5)
lambda_df <- map_df(lambdas, function(l) {
  mu <- mean(edx$rating)
  movie_effect <- edx %>%
```

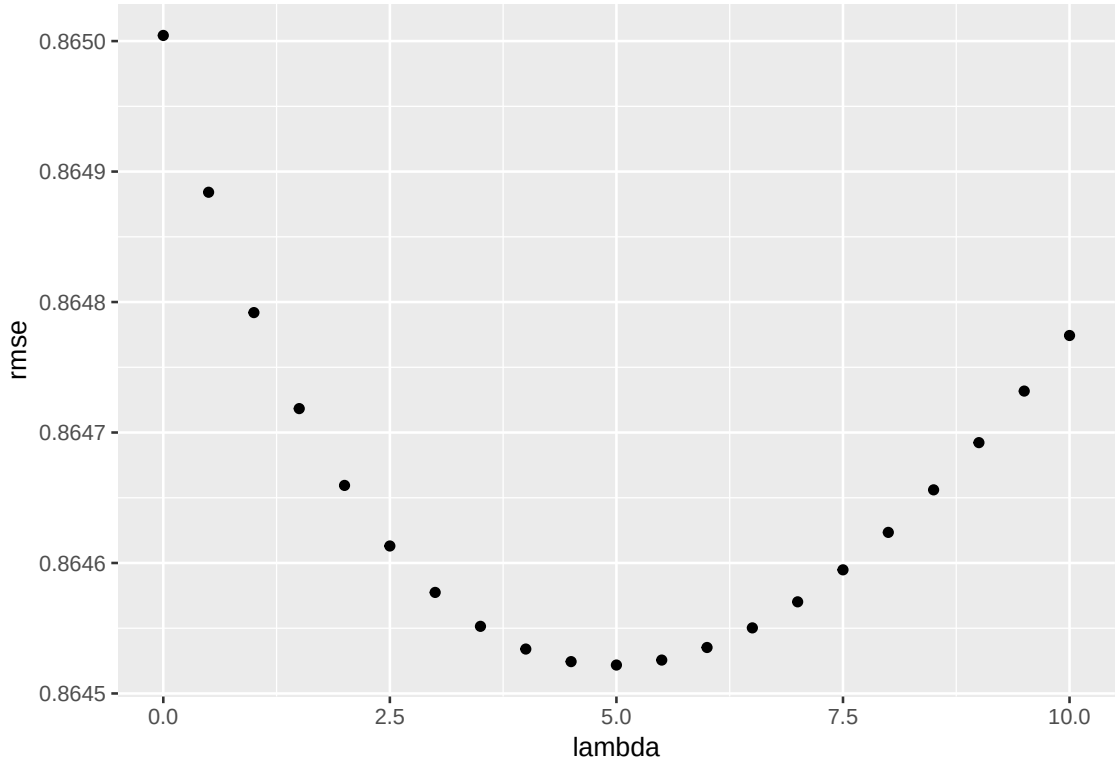
```
    group_by(movieId) %>%
    summarize(b_movie = sum(rating - mu)/(n() + l))
  user_effect <- edx %>%
    left_join(movie_effect, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - mu - b_movie)/(n() + l))
  year_effect <- edx %>%
    mutate(year_movie = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
    left_join(movie_effect, by = 'movieId') %>%
    left_join(user_effect, by = 'userId') %>%
    group_by(year_movie) %>%
    summarise(b_year = sum(rating - mu - b_movie - b_user)/(n() + l))

  predicted_real_ratings <- validation %>%
    mutate(year_movie = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
    left_join(movie_effect, by = 'movieId') %>%
    left_join(user_effect, by = 'userId') %>%
    left_join(year_effect, by = 'year_movie') %>%
    mutate(predicted_real = mu + b_movie + b_user + b_year) %>%
    .$predicted_real

  rmse <- RMSE(true_ratings = validation$rating, predicted_ratings = predicted_real_ratings)
  tibble(lambda =l, rmse = rmse)
})

lambda_best <- lambda_df$lambda[which.min(lambda_df$rmse)]

ggplot(lambda_df, aes(x = lambda, y = rmse)) + geom_point()
```



The best λ is equal to 5 and it can be so used to produce predictione and evaluating the model.

```
l <- lambda_best
mu <- mean(edx$rating)
movie_effect <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = sum(rating - mu)/(n() + l))
user_effect <- edx %>%
  left_join(movie_effect, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_user = sum(rating - mu - b_movie)/(n() + l))
year_effect <- edx %>%
  mutate(year_movie = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
  left_join(movie_effect, by = 'movieId') %>%
  left_join(user_effect, by = 'userId') %>%
  group_by(year_movie) %>%
  summarise(b_year = sum(rating - mu - b_movie - b_user)/(n() + l))

predicted_real_ratings <- validation %>%
  mutate(year_movie = as.numeric(str_sub(str_extract(title, '[0-9]{4}\\)$'), 1, 4))) %>%
  left_join(movie_effect, by = 'movieId') %>%
  left_join(user_effect, by = 'userId') %>%
  left_join(year_effect, by = 'year_movie') %>%
  mutate(predicted_real = mu + b_movie + b_user + b_year) %>%
  .$predicted_real

# evaluating root mean squared rror
rmse_1 <- RMSE(true_ratings = validation$rating, predicted_ratings = predicted_real_ratings)
rmse_results <- bind_rows(rmse_results,
```

```
                          tibble(method = 'movie + user + movie_year effects regularized',
                                 RMSE = rmse_1))

# remove objects to free memory
rm(movie_effect, user_effect, year_effect, lambdas, lambda_df)
gc()
```

Modeling latent factor reconstruction

In order to apply matrix factorization modeling, the rating matrix is created using the spread() function from tidyr package.

```
user_movie <- edx %>% select(userId, movieId, rating) %>%
  spread(key = movieId, value = rating)
userId_vec <- user_movie$userId
user_movie <- user_movie %>% select(-userId)
movieId_vec <- as.numeric(colnames(user_movie))
um_matrix <- as.matrix(user_movie)
```

The rating matrix created is full of NA. Matrix factorization PCA or SVD algorithm needs complete matrix and cannot be used with sparse matrix. softImpute implements an alternate least square algorithm that allow the factorization also in presence of a matrix with high rate of sparsity.

```
if(!require(softImpute)) {
  install.packages("softImpute", repos = "http://cran.us.r-project.org")
  library(softImpute)
}
um_matrix_sparse <- as(um_matrix, 'Incomplete')
rm(um_matrix, user_movie); gc()
```

The incomplete matrix object is now ready to be scaled by row (user) and column (movie) and factorized in 50 latent factors with 'als' algorithm. Note that lambda regularizing parameter has to be chosen so that the rank is 50 (It has been done through a parameter search but not reported in script for execution time constraint).

```
um_matrix_sparse_centered <- biScale(um_matrix_sparse,
                                     col.scale=FALSE, row.scale=FALSE,
                                     maxit = 50, thresh = 1e-05, trace=TRUE)
rating_fits <- softImpute(um_matrix_sparse_centered, type = "als",
                          rank.max = 51, lambda = 96,
                          trace=TRUE)
rating_fits$d
```

Note that lambda regularizing parameter has to be chosen so that the rank is 50. It is now possible to recontruct the matrix for validaion observation.

```
idx_user <- numeric(length = nrow(validation))
idx_movie <- numeric(length = nrow(validation))

for (it in 1:nrow(validation)) {
```

```
    idx_user[it] <- which(userId_vec == validation$userId[it])
    idx_movie[it] <- which(movieId_vec == validation$movieId[it])
}

latent_factor_effect <- numeric(length = length(idx_user))
latent_factor_effect <- impute(object = rating_fits,
                               i = idx_user , j = idx_movie,
                               unscale = FALSE)
# remove objects to free memory
rm(um_matrix_sparse, um_matrix_sparse_centered, idx_movie, idx_user)
gc()
```

Finally we add the effect calculated with matrix factorization to the previous regularized model.

```
# predict real ratings adding latent factor effect
predicted_real_ratings_mf <- predicted_real_ratings + latent_factor_effect


# evaluating root mean squared error
rmse_2 <- RMSE(true_ratings = validation$rating, predicted_ratings = predicted_real_ratings_mf)
rmse_results <- bind_rows(rmse_results,
                          tibble(method = 'matrix factorization',
                                 RMSE = rmse_2))
```

Results

The following table showed the results achieved in the 3 steps of the additive model.

method	RMSE
overall mean model	1.0612018
movie + user + movie_year effects	0.8650043
movie + user + movie_year effects regularized	0.8645218
matrix factorization	0.8231138

The final RMSE is below the null model rmse by 22% and also below the threshold set by HarvardX course Team to achieve 25 points. Every step in model building make the prediction better and better.

Conclusions

Interpreting the results, it is possible to say therefore that using this model on average the user rating for a specific movie can be far from the actual rating by 0.8231138. So the response to main project question about predicatbility is that it is possible to predict the rating that a particular user will give to a specific movie with 0.8231138 error on average.

Results can be considered valid because of this 3 main reasons:

- a consistent training / validation study design has been used (exception for tuning regularization parameter with the validation set)

- the training set and the validation set have huge number of observations (9000055 and 999999 respectively)
- the model theory is solid and tested (e.g. Netflix Prize)

This project helped me in understanding the data science research methodology and the expert use of statistical computation tool. Furthermore make me recall my former study in linear algebra as a student of electrical engineer years ago and understanding its application in the context of data science.

Model improvements

Future research should look at introducing more time effects with respect with timestamp variable not taken into account in this study. It should be interesting also evaluate the prediction score increasing by the number of latent factors (maybe up to 100). This should be done through validation. Furthermore we can explore the other algorithm for matrix completion of matrix provided by the softImpute package. As far as script performance is concerned for sure there is room for improvement in recoding for loops: options are parallelizing for loop or use Rcpp function to speed up the execution.

Reproducibilty

R script and rmarkdown file are available for review on public github repository:

https://github.com/mdt-ds/PH125_9x_MovieLens .

R script is intended to be reproducible. All package loading is checked for package installation. Directoty are all indicated in relative fashion. Furthermore in order to facilitate reproducibility, HW and SW used for this project have been reported below.

Computation

Computation capacity is a critical point: enough RAM shall be provided in order to perform the creation of the rating matrix and successive elaborations. On a laptop with 8 cores and 8 GB of RAM the following error is displayed when starting the third modeling step while trying to create the user_movie matrix: *Error: cannot allocate vector of size 2.8 Gb*. For this reason I have commented out this part on the script in order to allow running the script without error if memory of your computer is not enough. The script commented out runs in about 10 minutes on my laptop. Also on a Linux EC2 instance r5.large (2 vCPU, 16 GB) a similar error is displayed. Finally on a r5.xlarge (4 vCPU, 32 GB) EC2 instance the code has been executed without any error in about 16 minutes. In order to knit this report I had to use more powerful EC2 instance (r5.2xlarge with 8 vCPUs and 64 GB).

HW

The complete computation has been performed on a AWS EC2 instance of r5.xlarge type memory optimized: 4 virtual CPU and 32 GB RAM.

SW

The software stack is shown below launching sessionInfo() R function.

```
R version 3.5.2 (2018-12-20)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.1 LTS

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] lubridate_1.7.4 bindrcpp_0.2.2  caret_6.0-81    lattice_0.20-38
 [5] forcats_0.3.0   stringr_1.3.1   dplyr_0.7.8     purrr_0.2.5
 [9] readr_1.3.1     tidyr_0.8.2     tibble_2.0.0    ggplot2_3.1.0
[13] tidyverse_1.2.1

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.0          class_7.3-15        utf8_1.1.4
 [4] assertthat_0.2.0    digest_0.6.18       ipred_0.9-8
 [7] foreach_1.4.4       R6_2.3.0            cellranger_1.1.0
[10] plyr_1.8.4          backports_1.1.3     stats4_3.5.2
[13] evaluate_0.12       highr_0.7           httr_1.4.0
[16] pillar_1.3.1        rlang_0.3.1         lazyeval_0.2.1
[19] readxl_1.2.0        rstudioapi_0.8      data.table_1.11.8
[22] rpart_4.1-13        Matrix_1.2-15       rmarkdown_1.11
[25] labeling_0.3        splines_3.5.2       gower_0.1.2
[28] munsell_0.5.0       broom_0.5.1         compiler_3.5.2
[31] modelr_0.1.2        xfun_0.6            pkgconfig_2.0.2
[34] htmltools_0.3.6     nnet_7.3-12         tidyselect_0.2.5
[37] prodlim_2018.04.18 codetools_0.2-16    fansi_0.4.0
[40] crayon_1.3.4        withr_2.1.2         MASS_7.3-51.1
[43] recipes_0.1.4       ModelMetrics_1.2.2 grid_3.5.2
[46] nlme_3.1-137        jsonlite_1.6        gtable_0.2.0
[49] magrittr_1.5        scales_1.0.0        cli_1.0.1
[52] stringi_1.2.4       reshape2_1.4.3      timeDate_3043.102
[55] xml2_1.2.0          generics_0.0.2      lava_1.6.4
[58] iterators_1.0.10    tools_3.5.2         glue_1.3.0
[61] hms_0.4.2           survival_2.43-3     yaml_2.2.0
[64] colorspace_1.3-2    rvest_0.3.2         knitr_1.21
[67] bindr_0.1.1         haven_2.0.0
```

References

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[2] Rafael Irizarry (2018). Introduction to Data Science. Data Analysis and Prediction Algorithms with R. Chapters 71, 72 and 73 https://rafalab.github.io/dsbook/

[3] Edwin Chen (2011) Winning the Netflix Prize: A Summary. http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

[4] Trevor Hastie and Rahul Mazumder (2015). softImpute: Matrix Completion via Iterative Soft-Thresholded SVD. R package version 1.4. https://CRAN.R-project.org/package=softImpute

---