edX Capstone Movielens Project

Ciro B Rosa

27-Sep-2021

Introduction and Objective

This report is the result of the job performed on the "Movielens" dataset. The objective is to design a machine learning model that predicts movie ratings for a given user that has not previously seen that movie, based on data such as previous ratings to other movies, user's preferences, etc. The level of efficiency of the model is measured as RMSE (Root Mean Square Error), which means the lowest it the best.

Before you Begin: Prepare the Environment

This code has been tested on a specific Anaconda environment created on a Ubuntu 20.04 Linux Mate machine. All necessary scripts can be downloaded from the student's GitHub:

- https://github.com/cirobr/ds9-capstone-movielens.git

Next, the user should download and install Anaconda:

- https://www.anaconda.com/products/individual-d

Now, it is time to create an environment named "r-gpu" with the help of two command lines typed on terminal:

- conda create –name r-gpu python=3.9 notebook r-base=4.1 r-essentials r-e1071 r-irkernel r-varhandle r-foreach r-doparallel r-reticulate r-keras r-tfdatasets
- Rscript install-keras-gpu.R

Lastly, the below script run on RStudio executes the code provided by edX to generate the datasets and stores the files "edx.csv" and "validation.csv" on a sub-folder "./dat", which is also created in the process:

- code-preset.R

At this point, the code is ready for execution on RStudio, through this script:

- code-movielens.R

Code Organization and Project Development.

The code is developed in such a way as to execute the following tasks:

- Setup the environment, libraries and define key functions, such as to calculate RMSE;
- Read the edX dataset and split it on trainset / testset;
- Create, train and evaluate the model (naive average and neural network);
- Validate the model with the "validation" data set, and present the results.

The report will present all relevant outputs, as appropriate, in order to evidence the steps taken.

Project Development

```
# environment
library(reticulate)                         # interface R / Python
use_condaenv("r-gpu", required = TRUE)      # conda env for running tf and keras on gpu

# libraries
# library(stringi)                          # used on the code as stringi::stri_sub()
library(ggplot2)
library(lubridate)
```

Setup environment

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tidyverse)
```

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

## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

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

```
library(foreach)                            # multi-core computing for nzv()
```

```
##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
library(keras)                              # tensorflow wrap
library(tfdatasets)

# global variables
```

```
numberOfDigits <- 5
options(digits = numberOfDigits)
proportionTestSet <- 0.20
numberOfEpochs    <- 20                    # keras training parameter

# error function
errRMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

# function: difference between timestamps in days
daysBetweenTimestamps <- function(x,y){
  difftime(as_date(as_datetime(x)),
           as_date(as_datetime(y)),
           units = c("days")) %>%
    as.numeric()
}

# function: extract each genre from column genres
extractGenresNames <- function(elementVector){
  as.numeric(grepl(elementVector, genresVector))
}
```

Read and pre-process the edX dataset In order to ensure the "validation" data set is not handled at all at the training stage, it is deleted from memory. Please recall that its correspondent CSV file is stored on hard drive. Next, the "edx" data set is loaded to memory.

```
# clean memory
if(exists("validation")) {rm(validation)}

# read dataset from csv
print("pre-process edx")
```

```
## [1] "pre-process edx"
```

```
if(!exists("edx")) {edx <- read_csv(file = "./dat/edx.csv") %>% as_tibble()}
```

```
## Rows: 9000055 Columns: 6
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): title, genres
## dbl (4): userId, movieId, rating, timestamp
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(edx)
```

```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                         genres
##    <dbl>   <dbl>  <dbl>     <dbl> <chr>                         <chr>
## 1      1     122      5 838985046 Boomerang (1992)              Comedy|Romance
## 2      1     185      5 838983525 Net, The (1995)               Action|Crime|Th~
## 3      1     292      5 838983421 Outbreak (1995)               Action|Drama|Sc~
## 4      1     316      5 838983392 Stargate (1994)               Action|Adventur~
## 5      1     329      5 838983392 Star Trek: Generations (1994) Action|Adventur~
## 6      1     355      5 838984474 Flintstones, The (1994)       Children|Comedy~
```

Next, the following predictors will be extracted from the edX data set:

- "yearsFromRelease" is a predictor that indicates the number of years between film release and timestamp of evaluation. The year of release of the film is extracted as "yearOfRelease" from the "title" column.
- "daysFromFirstUserRating" is a predictor that indicates the number of days between the first assessment from a given user and the timestamp of assessment made by the same user. The first assessment from each user is a temporary variable.
- "daysFromFirstMovieRating" is similar to the above predictor. It gives the number of days between the first assessment a given movie has received, and the timestamp of the evaluation for that same movie. The first assessment granted for each movie is also a temporary variable.
- The column "genres" is a multiclass column that indicates the genre(s) of the movie that a given user has classified it. The code seeks for the available genres categories and creates a binary column for each of them.

```r
# move ratings to first column
edx2 <- edx %>% select(-c(rating))
edx2 <- cbind(rating = edx$rating, edx2)

# extract yearsFromRelease
edx2 <- edx2 %>%
  select(-c(genres)) %>%
  mutate(yearOfRelease = as.numeric(stringi::stri_sub(edx$title[1], -5, -2)),
         timestampYear = year(as_datetime(timestamp)),
         yearsFromRelease = timestampYear - yearOfRelease) %>%
  select(-c(title, yearOfRelease, timestampYear))

# extract firstUserRating
dfFirstUserRating <- edx2 %>% group_by(userId) %>%
  select(userId, timestamp) %>%
  summarize(firstUserRating = min(timestamp))

edx2 <- left_join(edx2, dfFirstUserRating)
```

```
## Joining, by = "userId"
```

```r
# extract firstMovieRating
dfFirstMovieRating <- edx2 %>% group_by(movieId) %>%
  select(movieId, timestamp) %>%
  summarize(firstMovieRating = min(timestamp))

edx2 <- left_join(edx2, dfFirstMovieRating)
```

```
## Joining, by = "movieId"
```

```r
# extract daysFromFirstUserRating and daysFromFirstMovieRating
edx2 <- edx2 %>% mutate(daysFromFirstUserRating  = daysBetweenTimestamps(timestamp, firstUserRating),
                        daysFromFirstMovieRating = daysBetweenTimestamps(timestamp, firstMovieRating)) %
  select(-c(timestamp, firstUserRating,firstMovieRating))

# extract movie genres as predictors
genresNames <- strsplit(edx$genres, "|", fixed = TRUE) %>%
  unlist() %>%
  unique()
```

```
genresVector <- edx$genres
df <- sapply(genresNames, extractGenresNames) %>% as_tibble()

# remove hyphen from predictor names
colnames(df)[7]  <- "SciFi"
colnames(df)[16] <- "FilmNoir"
colnames(df)[20] <- "NoGenre"

edx2 <- bind_cols(edx2, df)
head(edx2)
```

```
##   rating userId movieId yearsFromRelease daysFromFirstUserRating
## 1      5      1     122                4                       0
## 2      5      1     185                4                       0
## 3      5      1     292                4                       0
## 4      5      1     316                4                       0
## 5      5      1     329                4                       0
## 6      5      1     355                4                       0
##   daysFromFirstMovieRating Comedy Romance Action Crime Thriller Drama SciFi
## 1                      123      1       1      0     0        0     0     0
## 2                      153      0       0      1     1        1     0     0
## 3                      150      0       0      1     0        1     1     1
## 4                      154      0       0      1     0        0     0     1
## 5                      154      0       0      1     0        0     1     1
## 6                      154      1       0      0     0        0     0     0
##   Adventure Children Fantasy War Animation Musical Western Mystery FilmNoir
## 1         0        0       0   0         0       0       0       0        0
## 2         0        0       0   0         0       0       0       0        0
## 3         0        0       0   0         0       0       0       0        0
## 4         1        0       0   0         0       0       0       0        0
## 5         1        0       0   0         0       0       0       0        0
## 6         0        1       1   0         0       0       0       0        0
##   Horror Documentary IMAX NoGenre
## 1      0           0    0       0
## 2      0           0    0       0
## 3      0           0    0       0
## 4      0           0    0       0
## 5      0           0    0       0
## 6      0           0    0       0
```

```
# clean memory
rm(df, edx)
```

Split the edX dataset and check for stratification Next, the "edX" data set is split on trainset and testset. The resulting tables are then verified for its correct stratification, with the aid of a chart that demonstrates the data splitting has also split each movie rating category, ranging between [0.5; 5.0], at approximately the same 80% trainset / 20% testset proportion.

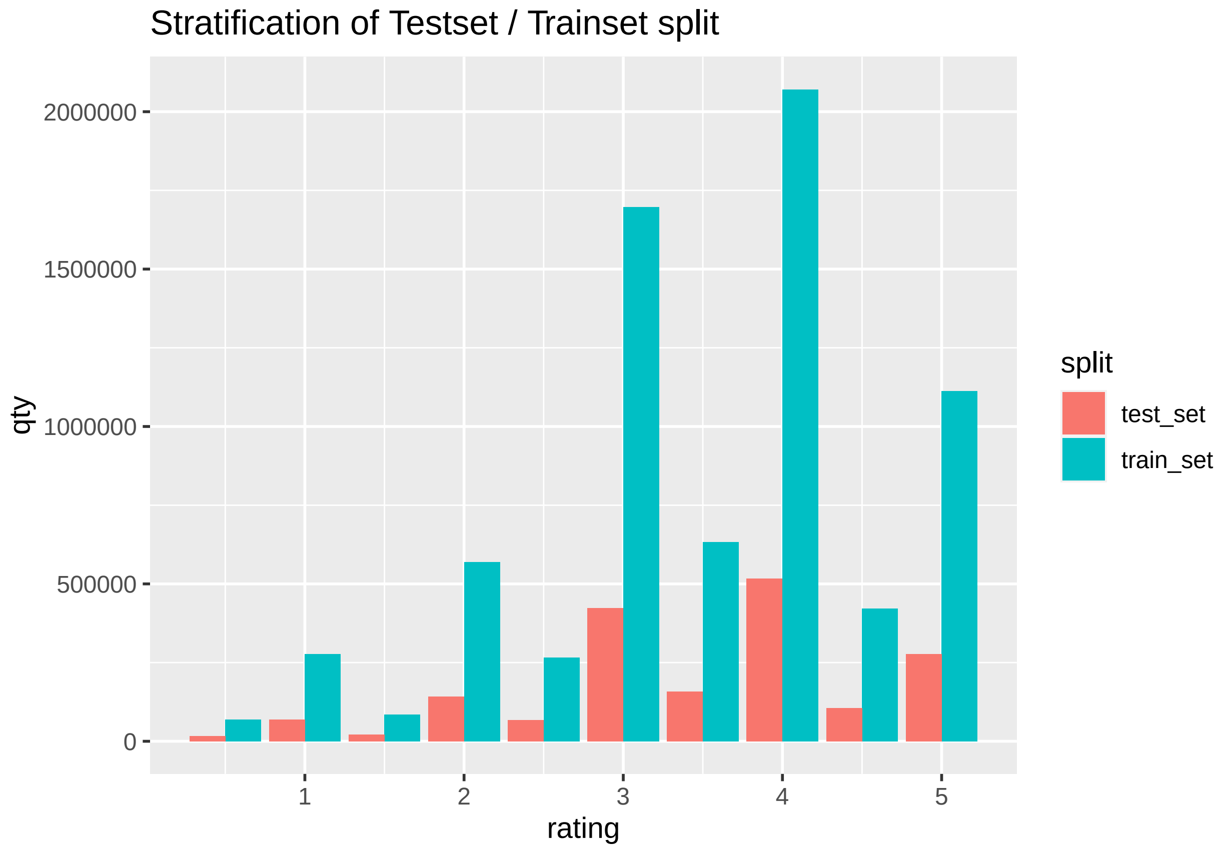
```
# split train and test sets
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(edx2$rating,
                                  times = 1,
                                  p = proportionTestSet,
                                  list = FALSE)
```

```
test_set <- edx2 %>% slice(test_index)
train_set <- edx2 %>% slice(-test_index)

# check for stratification of train / test split
p1 <- train_set %>%
  group_by(rating) %>%
  summarize(qty = n()) %>%
  mutate(split = 'train_set')

p2 <- test_set %>%
  group_by(rating) %>%
  summarize(qty = n()) %>%
  mutate(split = 'test_set')

p <- bind_rows(p1, p2) %>% group_by(split)
p %>% ggplot(aes(rating, qty, fill = split)) +
  geom_bar(stat="identity", position = "dodge") +
  ggtitle("Stratification of Testset / Trainset split")
```



Pre processing of trainset The trainset is now pre processed for dimensionality reduction by eliminating the small variance predictors. This step is important as it will reduce computational workload at the neural network processing steps.

```
# remove movies and users from testset that are not present on trainset
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# remove predictors with small variance
nzv <- train_set %>%
  select(-rating) %>%
  nearZeroVar(foreach = TRUE, allowParallel = TRUE)
removedPredictors <- colnames(train_set[,nzv])
removedPredictors
```

```
## [1] "Animation"   "Musical"     "Mystery"     "Horror"      "Documentary"
## [6] "IMAX"
```

```
train_set <- train_set %>% select(-all_of(removedPredictors))
test_set <- test_set %>% select(-all_of(removedPredictors))

# cleanup memory
rm(edx2, test_index)
rm(p, p1, p2)
```

First model: Naive Average Next, the naive average model is built as a baseline figure of merit for the project, which means that further processing is expected to, as a minimum, deliver performance better than achieved so far.

```
### predict by global average
mu <- mean(train_set$rating)
predicted <- mu
err <- RMSE(test_set$rating, predicted)
rmse_results <- tibble(model = "naiveAverage",
                       error = err)
rmse_results
```

```
## # A tibble: 1 x 2
##   model        error
##   <chr>        <dbl>
## 1 naiveAverage  1.06
```

Dataset preparation for neural network processing: The below code is a pre-processing of data for the forthcoming neural network processing. The following steps are taken:

- A “movie bias” and “user bias” indexes are extracted, then merged to the train/test sets. The idea of extracting such features might allow the model to capture the “taste” of users to e.g. blockbusters, among others.

```
# add movie bias effect
dfBiasMovie <- train_set %>%
  select(rating, movieId) %>%
  group_by(movieId) %>%
  summarize(biasMovie = mean(rating))
head(dfBiasMovie)
```

```
## # A tibble: 6 x 2
##   movieId biasMovie
##     <dbl>     <dbl>
## 1       1      3.93
## 2       2      3.20
## 3       3      3.14
## 4       4      2.87
## 5       5      3.06
## 6       6      3.82
```

```r
# add user bias effect
dfBiasUser <- train_set %>%
  select(rating, userId) %>%
  group_by(userId) %>%
  summarize(biasUser = mean(rating))
head(dfBiasUser)
```

```
## # A tibble: 6 x 2
##   userId biasUser
##    <dbl>    <dbl>
## 1      1     5
## 2      2     3.08
## 3      3     3.94
## 4      4     4.14
## 5      5     4.11
## 6      6     4
```

```r
df_train <- train_set %>%
  left_join(dfBiasMovie) %>%
  left_join(dfBiasUser) %>%
  as_tibble()
```

```
## Joining, by = "movieId"
```

```
## Joining, by = "userId"
```

```r
df_test <- test_set %>%
  left_join(dfBiasMovie) %>%
  left_join(dfBiasUser) %>%
  as_tibble()
```

```
## Joining, by = "movieId"
## Joining, by = "userId"
```

```r
head(df_train)
```

```
## # A tibble: 6 x 22
##   rating userId movieId yearsFromRelease daysFromFirstUserR~ daysFromFirstMovie~
##    <dbl>  <dbl>   <dbl>            <dbl>               <dbl>               <dbl>
## 1      5      1     185                4                   0                 153
## 2      5      1     316                4                   0                 154
## 3      5      1     329                4                   0                 154
## 4      5      1     355                4                   0                 154
## 5      5      1     364                4                   0                 154
## 6      5      1     377                4                   0                 154
## # ... with 16 more variables: Comedy <dbl>, Romance <dbl>, Action <dbl>,
## #   Crime <dbl>, Thriller <dbl>, Drama <dbl>, SciFi <dbl>, Adventure <dbl>,
## #   Children <dbl>, Fantasy <dbl>, War <dbl>, Western <dbl>, FilmNoir <dbl>,
## #   NoGenre <dbl>, biasMovie <dbl>, biasUser <dbl>
```

```r
# clean memory
rm(train_set, test_set)
```

Second model: Neural Network The code presented next takes the necessary steps to configure, compile, train and test a Neural Network model. This student has chosen to go through this way as a novel approach, in the sense that it has not been exploited at all in classes. An excellent online lecture about the theory behind neural networks can be found here:

- https://youtu.be/Ih5Mr93E-2c

The baseline package used for training the model is "Keras", which is a wrap for "Tensorflow". The technical reference for programming with the package is found at the following links:

- https://cran.r-project.org/web/packages/keras/vignettes/index.html
- https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_regression/
- https://datascience.stackexchange.com/questions/57171/how-to-improve-low-accuracy-keras-model-design/57292

The Keras package offers a variety of activation functions for its neurons. However, as the package may consume a significant amount of computer resources, this project will focus only on "Relu" activation function, and will not conduct a grid search among several activation functions.

Please note that the code can take +2h before it ends at an relatively usual Intel core i7 machine with GPU. At the end, the validation of result on each epoch is presented on a chart:

```
# scale predictors
spec <- feature_spec(df_train, rating ~ . ) %>%
  step_numeric_column(all_numeric(), normalizer_fn = scaler_standard()) %>%
  fit()
spec
```

```
## -- Feature Spec ----------------------------------------------------------------
## A feature_spec with 21 steps.
## Fitted: TRUE
## -- Steps -----------------------------------------------------------------------
## The feature_spec has 1 dense features.
## StepNumericColumn: userId, movieId, yearsFromRelease, daysFromFirstUserRating, daysFromFirstMovieRat
## -- Dense features --------------------------------------------------------------
```

```
# wrap the model in a function
build_model <- function() {
  # create model
  input <- layer_input_from_dataset(df_train %>% select(-c(rating)))

  output <- input %>%
    layer_dense_features(dense_features(spec)) %>%
    layer_dense(units = 32, activation = "relu") %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 8, activation = "relu") %>%
    layer_dense(units = 8, activation = "relu") %>%
    layer_dense(units = 1)

  model <- keras_model(input, output)
  summary(model)

  # compile model
  model %>%
    compile(
      loss = "mse",
      optimizer = optimizer_rmsprop(),
      metrics = list("mean_absolute_error")
    )

  model
}

# train the model
print_dot_callback <- callback_lambda(
  on_epoch_end = function(epoch, logs) {
    if (epoch %% 80 == 0) cat("\n")
    cat(".")
  }
)

early_stop <- callback_early_stopping(monitor   = "val_loss",
                                      min_delta = 1e-5,
                                      patience  = 5,
                                      mode      = "min",
                                      restore_best_weights = TRUE)
model <- build_model()
```

```
## Model: "model"
## ________________________________________________________________________________
## Layer (type)              Output Shape      Param #  Connected to
## ================================================================================
## Action (InputLayer)       [(None,)]         0
## ________________________________________________________________________________
## Adventure (InputLayer)    [(None,)]         0
## ________________________________________________________________________________
## Children (InputLayer)     [(None,)]         0
## ________________________________________________________________________________
## Comedy (InputLayer)       [(None,)]         0
## ________________________________________________________________________________
## Crime (InputLayer)        [(None,)]         0
## ________________________________________________________________________________
## Drama (InputLayer)        [(None,)]         0
## ________________________________________________________________________________
## Fantasy (InputLayer)      [(None,)]         0
## ________________________________________________________________________________
## FilmNoir (InputLayer)     [(None,)]         0
## ________________________________________________________________________________
## NoGenre (InputLayer)      [(None,)]         0
## ________________________________________________________________________________
## Romance (InputLayer)      [(None,)]         0
## ________________________________________________________________________________
## SciFi (InputLayer)        [(None,)]         0
## ________________________________________________________________________________
## Thriller (InputLayer)     [(None,)]         0
## ________________________________________________________________________________
## War (InputLayer)          [(None,)]         0
## ________________________________________________________________________________
## Western (InputLayer)      [(None,)]         0
## ________________________________________________________________________________
## biasMovie (InputLayer)    [(None,)]         0
## ________________________________________________________________________________
## biasUser (InputLayer)     [(None,)]         0
## ________________________________________________________________________________
## daysFromFirstMovieRating  [(None,)]         0
## ________________________________________________________________________________
## daysFromFirstUserRating ( [(None,)]         0
## ________________________________________________________________________________
## movieId (InputLayer)      [(None,)]         0
## ________________________________________________________________________________
## userId (InputLayer)       [(None,)]         0
## ________________________________________________________________________________
## yearsFromRelease (InputLa [(None,)]         0
## ________________________________________________________________________________
## dense_features (DenseFeat (None, 21)        0        Action[0][0]
##                                                      Adventure[0][0]
##                                                      Children[0][0]
##                                                      Comedy[0][0]
##                                                      Crime[0][0]
##                                                      Drama[0][0]
##                                                      Fantasy[0][0]
##                                                      FilmNoir[0][0]
##                                                      NoGenre[0][0]
##                                                      Romance[0][0]
##                                                      SciFi[0][0]
##                                                      Thriller[0][0]
##                                                      War[0][0]
##                                                      Western[0][0]
##                                                      biasMovie[0][0]
##                                                      biasUser[0][0]
##                                                      daysFromFirstMovieRating[0]
##                                                      daysFromFirstUserRating[0][
##                                                      movieId[0][0]
##                                                      userId[0][0]
##                                                      yearsFromRelease[0][0]
## ________________________________________________________________________________
## dense_5 (Dense)           (None, 32)        704      dense_features[0][0]
## ________________________________________________________________________________
## dense_4 (Dense)           (None, 16)        528      dense_5[0][0]
## ________________________________________________________________________________
## dense_3 (Dense)           (None, 16)        272      dense_4[0][0]
## ________________________________________________________________________________
## dense_2 (Dense)           (None, 8)         136      dense_3[0][0]
## ________________________________________________________________________________
## dense_1 (Dense)           (None, 8)         72       dense_2[0][0]
## ________________________________________________________________________________
## dense (Dense)             (None, 1)         9        dense_1[0][0]
## ================================================================================
## Total params: 1,721
## Trainable params: 1,721
## Non-trainable params: 0
## ________________________________________________________________________________
```

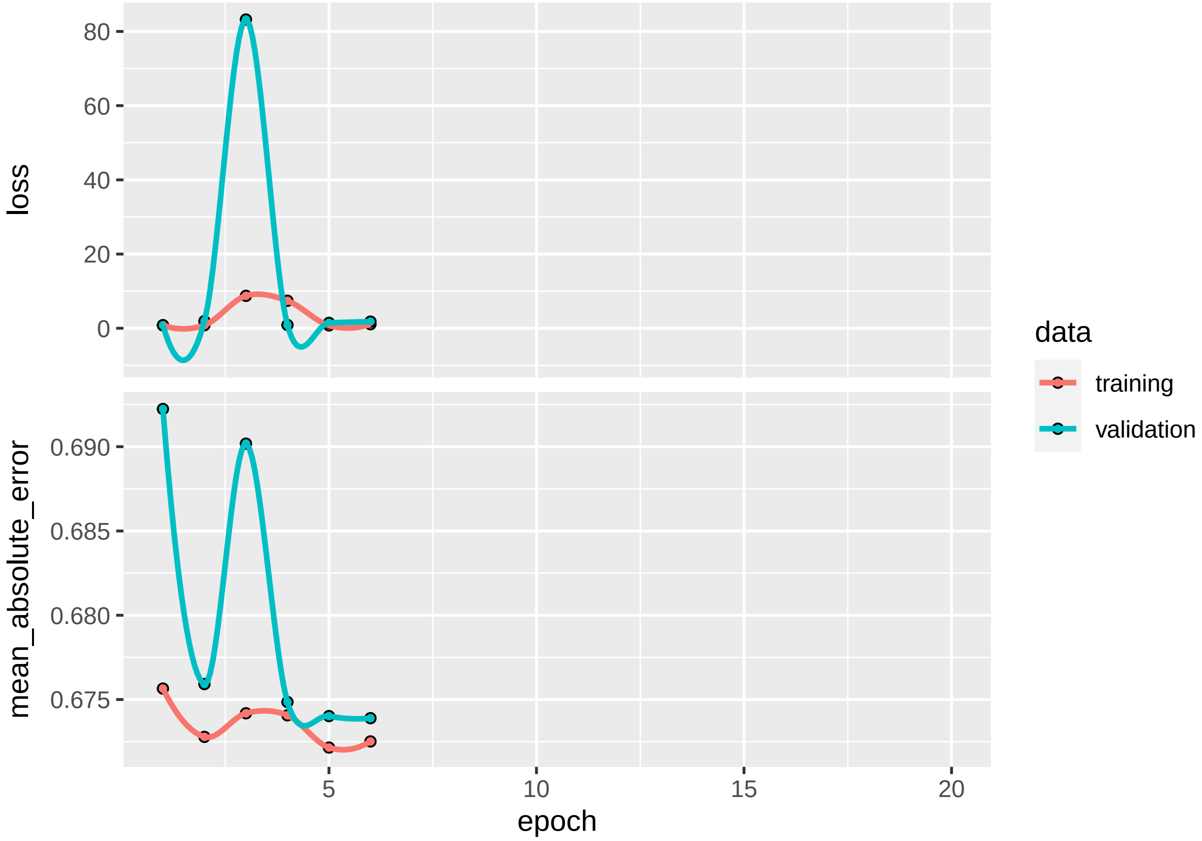
```
history <- model %>% fit(
  x = df_train %>% select(-c(rating)),
```

```r
  y = df_train$rating,
  epochs = numberOfEpochs,
  validation_split = 0.2,
  verbose = 0,
  callbacks = list(early_stop, print_dot_callback)
)
```

```
##
## ......
```

```r
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



The prediction on the validation set is now performed:

```r
# predict
predicted <- model %>% predict(df_test %>% select(-c(rating)))
predicted <- predicted[ , 1]

# calculate error metrics
err <- errRMSE(df_test$rating, predicted)

rmse_results <- bind_rows(rmse_results,
                          tibble(model ="CNN",
                                 error = err))
rmse_results
```

```
## # A tibble: 2 x 2
##   model        error
##   <chr>        <dbl>
## 1 naiveAverage 1.06
## 2 CNN          0.901
```

```
# clean memory
rm(df_train, df_test)
```

Validation: The Final Step Given that we have a tested model, it is time to validate it. The "validation" dataset is now recovered and pre-processed for use, then the predictions are made over it.

The validation set needs to be pre-processed before being used on predictions. This is accomplished in a similar way as for the trainset/testset before:

```
# read dataset from csv
validation <- read_csv(file = "./dat/validation.csv") %>% as_tibble()
```

```
## Rows: 999999 Columns: 6

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): title, genres
## dbl (4): userId, movieId, rating, timestamp

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(validation)
```

```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                      genres
##    <dbl>   <dbl>  <dbl>     <dbl> <chr>                      <chr>
## 1      1     231      5 838983392 Dumb & Dumber (1994)       Comedy
## 2      1     480      5 838983653 Jurassic Park (1993)       Action|Adventure|S~
## 3      1     586      5 838984068 Home Alone (1990)          Children|Comedy
## 4      2     151      3 868246450 Rob Roy (1995)             Action|Drama|Roman~
## 5      2     858      2 868245645 Godfather, The (1972)      Crime|Drama
## 6      2    1544      3 868245920 Lost World: Jurassic Park~ Action|Adventure|H~
```

```
# prepare validation dataset
df_val <- validation %>%
  select(-c(rating))
df_val <- cbind(rating = validation$rating, df_val)

df_val <- df_val %>%
  select(-c(genres)) %>%
  mutate(yearOfRelease = as.numeric(stringi::stri_sub(validation$title[1], -5, -2)),
         timestampYear = year(as_datetime(timestamp)),
         yearsFromRelease = timestampYear - yearOfRelease) %>%
  select(-c(title, yearOfRelease, timestampYear)) %>%
  left_join(dfFirstUserRating) %>%
  left_join(dfFirstMovieRating) %>%
  mutate(daysFromFirstUserRating  = daysBetweenTimestamps(timestamp, firstUserRating),
         daysFromFirstMovieRating = daysBetweenTimestamps(timestamp, firstMovieRating)) %>%
  select(-c(timestamp, firstUserRating,firstMovieRating))
```

```
## Joining, by = "userId"
```

```
## Joining, by = "movieId"
```

```
genresVector <- validation$genres
df <- sapply(genresNames, extractGenresNames) %>% as_tibble()
colnames(df)[7]  <- "SciFi"
colnames(df)[16] <- "FilmNoir"
colnames(df)[20] <- "NoGenre"
df_val <- bind_cols(df_val, df)

df_val <- df_val %>%
  select(-all_of(removedPredictors)) %>%
  left_join(dfBiasMovie) %>%
  left_join(dfBiasUser) %>%
  as_tibble()
```

```
## Joining, by = "movieId"
```

```
## Joining, by = "userId"
```

```
head(df_val)
```

```
## # A tibble: 6 x 22
##   rating userId movieId yearsFromRelease daysFromFirstUserR~ daysFromFirstMovie~
##    <dbl>  <dbl>   <dbl>            <dbl>               <dbl>               <dbl>
## 1      5      1     231                2                   0                 154
## 2      5      1     480                2                   0                 154
## 3      5      1     586                2                   0                  73
## 4      3      2     151                3                   0                 489
## 5      2      2     858                3                   0                 325
## 6      3      2    1544                3                   0                  46
## # ... with 16 more variables: Comedy <dbl>, Romance <dbl>, Action <dbl>,
## #   Crime <dbl>, Thriller <dbl>, Drama <dbl>, SciFi <dbl>, Adventure <dbl>,
## #   Children <dbl>, Fantasy <dbl>, War <dbl>, Western <dbl>, FilmNoir <dbl>,
## #   NoGenre <dbl>, biasMovie <dbl>, biasUser <dbl>
```

```
# predict
predicted <- model %>% predict(df_val %>% select(-c(rating)))
predicted <- predicted[ , 1]
validRows <- !is.na(predicted)

# calculate error metrics
err <- errRMSE(df_val$rating[validRows], predicted[validRows])

rmse_results <- bind_rows(rmse_results,
                          tibble(model ="CNN validation",
                                 error = err))
rmse_results
```

```
## # A tibble: 3 x 2
##   model          error
##   <chr>          <dbl>
## 1 naiveAverage   1.06
## 2 CNN            0.901
## 3 CNN validation 0.884
```

```
# clean memory
rm(df, df_val, validation, predicted, validRows)
```

Conclusions

The student has demonstrated he has learned several skills from the classes that enabled him to pre-process data and exploit by himself the topic of “neural networks”, that has not been covered during the course, and reaching to an RMSE of around 0.880, which is a significant improvement from the naive average approach.

From the validation results, one can observe its RMSE result is slightly higher than the one achieved with the testset. This is a good indication of the generalization of the model.

Next steps / Future work

The student plans to go further on studying the following topics in more details, in order to improve the result of this task and all future tasks: * KNN; * PCA and SVM; * Neural Networks and the Keras package; * Ensembles.