Report

Egar Garcia

1/22/2019

Overview

Methodology

To evaluate the different (machine learning) methods tested in this report, an object-oriented approach is going to be used, each method is going to be represented by a model which would be generated through the construction of an object. To generate a model, the constructor function receives a dataset (the training set) which is used to fit the model an construct its object. The model's object includes a `predict` function used to perform a prediction for another dataset (which could be the set for testing, validation, production, etc.).

For example, the following function creates and object to represent a model that always gives the most common rate (the mode) of the training set as prediction:

```
#' This object-constructor function is used to generate a model that returns
#' a as prediction the most common rating in the dataset used to fit it.
#' @param dataset The dataset used to fit the model
#' @return The model
ModeModel <- function(dataset) {
  model <- list()

  model$ratings <- unique(dataset$rating)
  model$mode <- model$ratings[which.max(tabulate(match(dataset$rating, model$ratings)))]

  #' The prediction function
  #' @param s The dataset used to perform the prediction of
  #' @return A vector containing the prediction for the given dataset
  model$predict <- function(s) {
    model$mode
  }

  model
}
```

Using the constructor function, an object can be created to fit the particular model:

```
model <- ModeModel(edx)
```

Then this model can be used to make predictions, like in the following code predicting agains the training and the validation sets:

```
training_pred <- model$predict(edx)
validation_pred <- model$predict(validation)
```

And the predictions are helpful to messure the performance of the model in terms of RMSE and/or accuracy, applied to the training and validation sets:

```
sprintf("Train-RMSE: %f, Train-Acc: %f, Val-RMSE: %f, Val-Acc: %f",
        RMSE(training_pred, edx$rating),
        mean(training_pred == edx$rating),
```

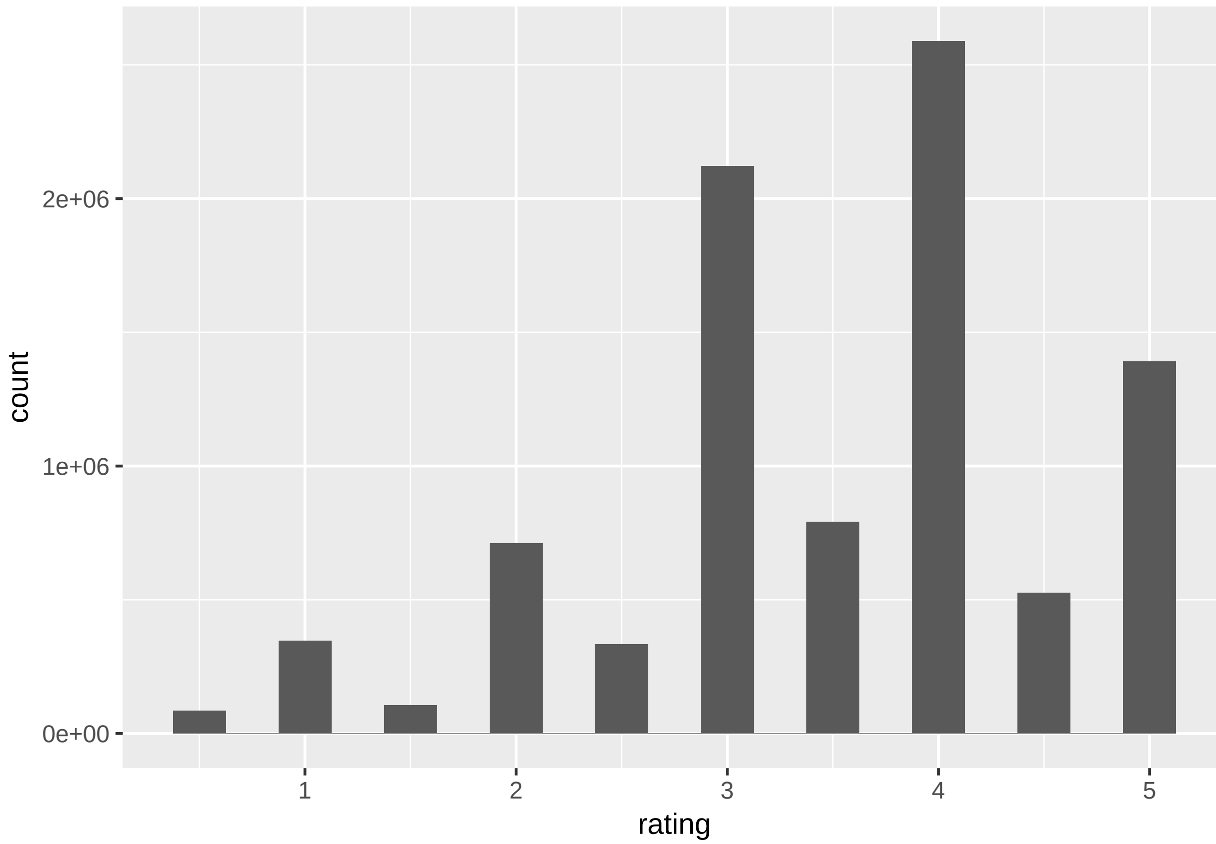
```r
        RMSE(validation_pred, validation$rating),
        mean(validation_pred == validation$rating))
```

```
## [1] "Train-RMSE: 1.167044, Train-Acc: 0.287602, Val-RMSE: 1.168016, Val-Acc: 0.287420"
```

Analysing the data

Let's take an initial look at how the ratings are distributed by plotting an histogram to account the amount of the different ratings given by the customers.

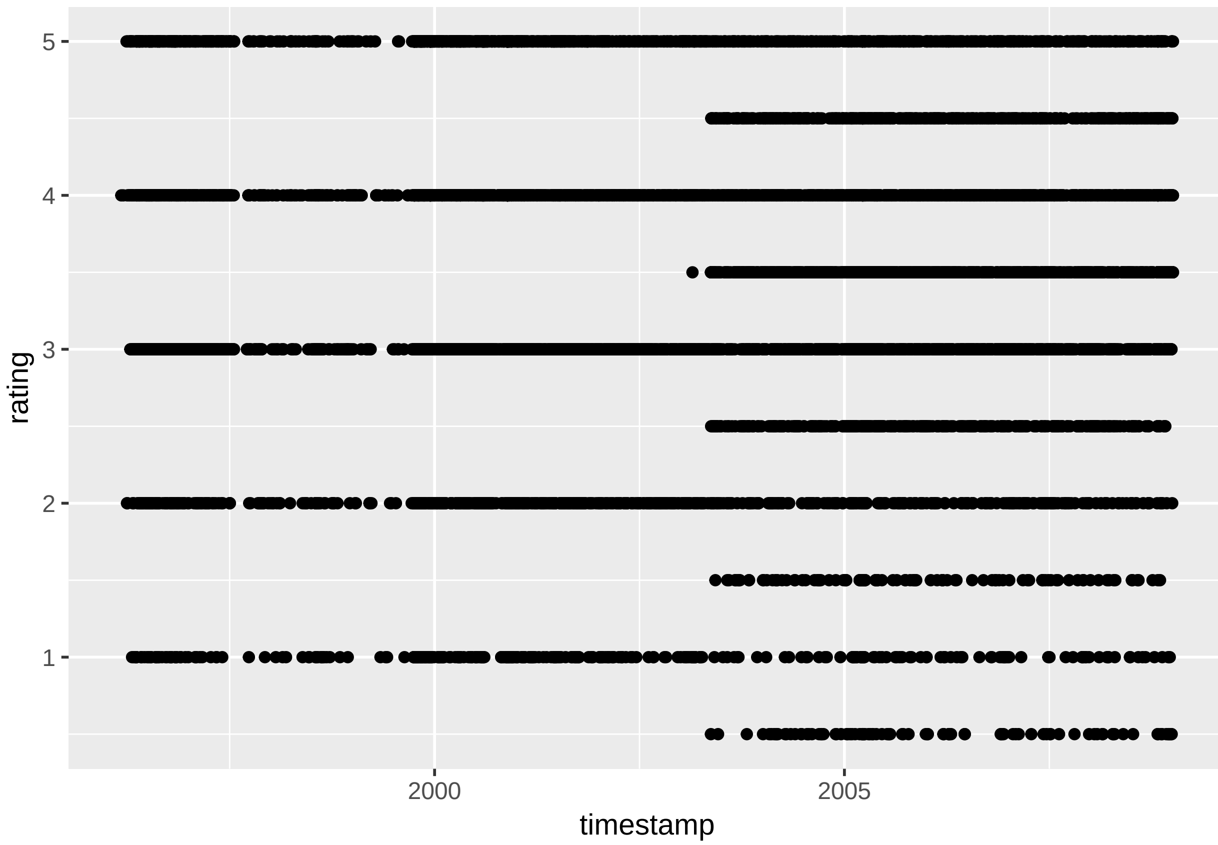
```r
edx %>%
  ggplot() +
  geom_histogram(aes(x = rating), binwidth = 0.25)
```



It can be observed that ratings can be interpreted as the number of starts from 1 to 5 that users give to a movie, however it can be seen that ratings ending with a half are used, at first impression looks like half start ratings are not very popular among users.

Let's visualize the data from another point of view, this time plotting the ratings against the timestamp. For exploratory purposes lets just plot using a small subset of the dataset, since using the whole one might take a while.

```r
edx[createDataPartition(y = edx$rating, times = 1, p = 0.001, list = FALSE),] %>%
  ggplot(aes(x = as_datetime(timestamp), y = rating)) +
  geom_point() +
  labs(x = 'timestamp', y = 'rating')
```



Now something interesting can be observed, it looks like ratings ending in half-stars were allowed after certain point in time, and before that just full-stars were allowed.

To find the point in time were rating ending in half star started to appear, the following code can be used. In basically gets the minimum timestamp in the dataset where a rating with half star is found.

```
half_stars_startpoint <- min(filter(edx, (rating * 2) %% 2 == 1)$timestamp)
```

Converting `half_stars_startpoint` to a more redable representation using the following code:

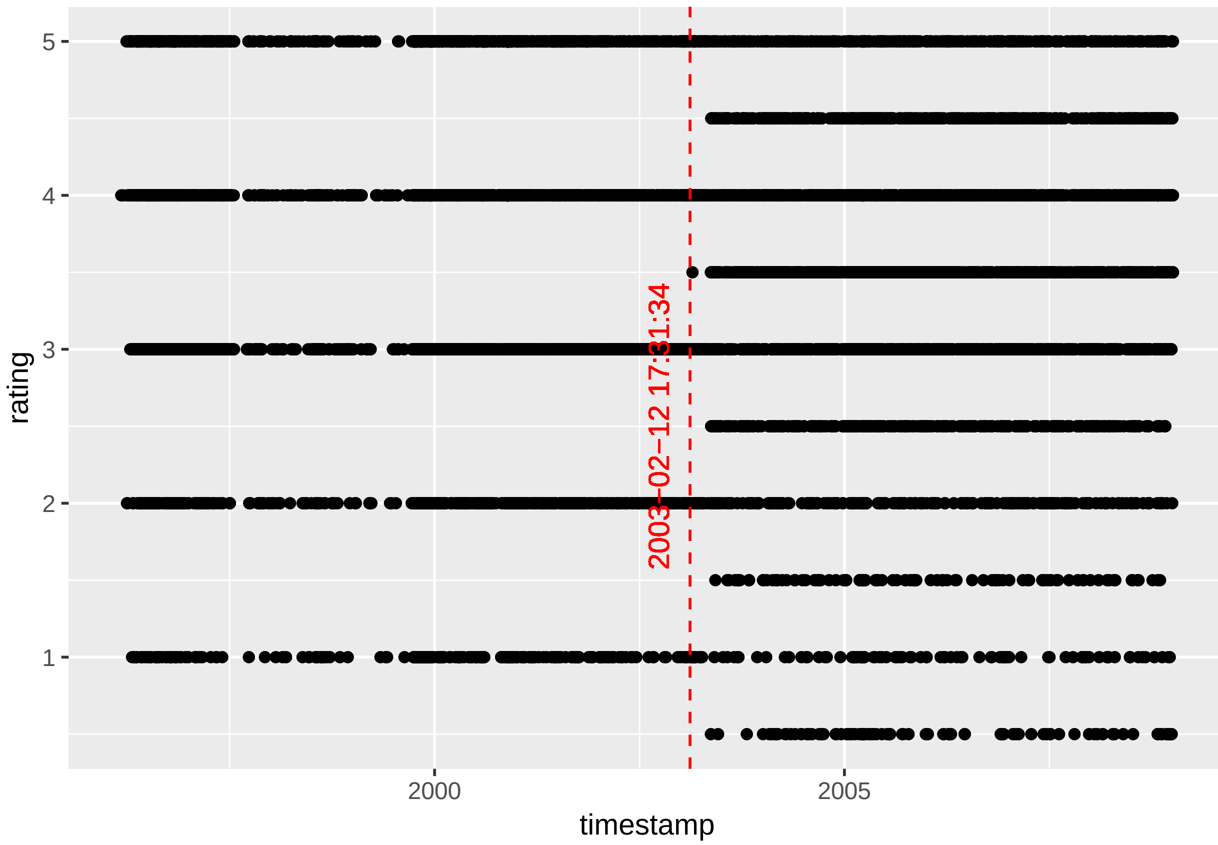
```
library(lubridate)
as_datetime(half_stars_startpoint)
```

```
## [1] "2003-02-12 17:31:34 UTC"
```

It can be seen that the point in time when half-star ratings started to appear was on 2003-02-12 17:31:34.

Again, let's plot the ratings against the timestamp, but this time adding a vertical line indicating the point in time where half-star ratings were allowed.

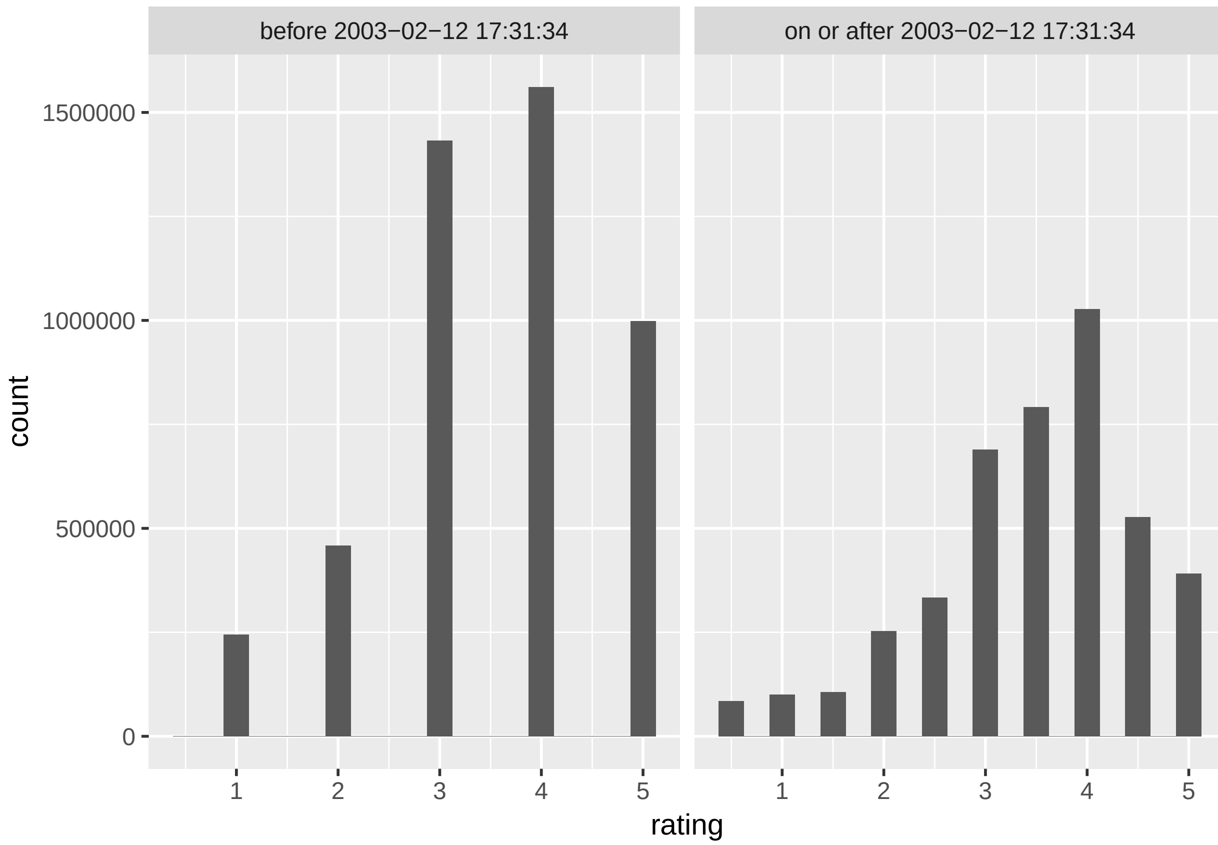
```
edx[createDataPartition(y = edx$rating, times = 1, p = 0.001, list = FALSE),] %>%
  ggplot(aes(x = as_datetime(timestamp), y = rating)) +
  geom_point() +
  geom_vline(aes(xintercept = as_datetime(half_stars_startpoint)),
             color = "red", linetype = "dashed") +
  geom_text(aes(x = as_datetime(half_stars_startpoint),
                label = as_datetime(half_stars_startpoint),
                y = 2.5),
            color = "red", vjust = -1, angle = 90) +
  labs(x = 'timestamp', y = 'rating')
```



A clear partition of the dataset can be observed, the first one for ratings before 2003-02-12 17:31:34 where only full-stars were allowed, and a second one for ratings after that point in time where half-stars were allowed. Let's create another plot about the distribution of ratings in each one of these partitions.

```
partition_names = c(paste('before', as_datetime(half_stars_startpoint)),
                    paste('on or after', as_datetime(half_stars_startpoint)))

edx %>%
  mutate(partition = factor(ifelse(timestamp < half_stars_startpoint,
                                   partition_names[1], partition_names[2]),
                            levels = partition_names)) %>%
  ggplot() +
  geom_histogram(aes(x = rating), binwidth = 0.25) +
  facet_grid(~ partition)
```



Now the distribution of ratings looks very different, it seems that ratings using half-stars were also popular among users when they were allowed (i.e. in the second partition).

Having a point in time to partition the dataset seems to be an important aspect to consider, since it could mean a different users behavior from one partition to another. For example, it might be that a partucular user had a tendency to rate movies with 4 stars before 2003-02-12 17:31:34, but then after that date the same user might have changed its tendency to rate 3.5 instead, since now half-stars were allowed.

Refining the metodology to use partitioned models

```
#' This object-constructor function is used to generate a metamodel
#' that contains two models,
#' one fitted for data before the startpoint when half stars were allowed in the
#' ratings, and the other one fitted for data on or after that startpoint.
#' The predictions are performed by choosing the appropriate model according to the
#' data's timestamp.
#'
#' @param dataset The dataset used to fit both models,
#'    it should contain a column called 'timestamp'
#' @param base_model_generator The function used to generate the base models,
#'    it should receive a dataset to fit the model and have a prediction function
#' @return The created metamodel
PartitionedModel <- function(dataset, base_model_generator) {
  partitioned_model <- list()

  # Spliting the dataset in 2,
```

```r
  # one set for data before the startpoint when half stars were allowed
  dataset1 <- dataset %>% filter(timestamp < half_stars_startpoint)
  # and the other one for the data on or after the startpoint when half stars were allowed
  dataset2 <- dataset %>% filter(timestamp >= half_stars_startpoint)

  # Generating a model for each dataset
  partitioned_model$model1 <- base_model_generator(dataset1)
  partitioned_model$model2 <- base_model_generator(dataset2)

  #' Performs a prediction with the combined fitted models,
  #' it tries to do the prediction with the respective model based on the timestamp,
  #' but if the prediction can not be performed then the other model is attempted.
  #' @param s The dataset used to perform the prediction of
  #' @return A vector containing the prediction for each row of the dataset
  partitioned_model$predict <- function(s) {
    # Performing the predictions on the whole dataset for each one of the models
    pred1 <- partitioned_model$model1$predict(s)
    pred2 <- partitioned_model$model2$predict(s)

    # Selecting the prediction to use according to the data's timestamp,
    # if a prediction is missing the prediction for the other model is used
    s %>%
      mutate(pred1 = pred1, pred2 = pred2) %>%
      mutate(pred = ifelse(timestamp < half_stars_startpoint,
                           ifelse(!is.na(pred1), pred1, pred2),
                           ifelse(!is.na(pred2), pred2, pred1))) %>%
      .$pred
  }

  partitioned_model
}
```

```r
#' Converts a prediction (which is a floating point number) to a one used to
#' represent ratings given by stars,
#' i.e. {1, 2, 3, 4, 5} if the timestamp is before the half start startpoint
#' or {1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5} if the timestamp is on or after.
pred2stars <- function(timestamp, pred) {
  # Rounds the prediction either to be full-stars or having a half-star
  # according to the timestamp
  rounded_pred <- ifelse(timestamp < half_stars_startpoint,
                         round(pred),
                         round(pred * 2)/2)

  # Making sure the rating is not smaller that 1 or bigger than 5
  min(max(rounded_pred, 1), 5)
}
```

```r
#' This function is used to report the performance of a model in terms of
#' RMSE and Accuracy for the training and validation sets.
#' It evaluates the performance in two modes:
#' 1) using the whole training set to fit the model and
#' 2) partitioning the training set before and on-or-after
#' the startpoint when half stars were allowed.
#' @param model_generator The constructor function to generate the model
```

```r
#' @returns A dataset reporting the performance results
get_performance_metrics <- function(model_generator) {
  dataset_names <- c('Training', 'Validation')
  datasets <- c()
  modes <- c()
  rmses <- c()
  accuracies <- c()
  counter <- 0

  for (is_partitioned in c(FALSE, TRUE)) {
    # Chosing the mode PARTITIONED or WHOLE
    if (is_partitioned) {
      model <- PartitionedModel(edx, model_generator)
    } else {
      model <- model_generator(edx)
    }

    for (dataset_name in dataset_names) {
      # Chosing the dataset to evaluate
      if (dataset_name == 'Training') {
        ds <- edx
      } else {
        ds <- validation
      }

      counter <- counter + 1

      # Getting the prediction for the chosen dataset
      pred <- model$predict(ds)

      datasets[counter] <- dataset_name
      modes[counter] <- ifelse(is_partitioned, 'PARTITIONED', 'WHOLE')

      # Calculating the RMSE
      rmses[counter] <- RMSE(pred, ds$rating)
      # Calculating the accuracy
      accuracies[counter] <- mean(pred2stars(ds$timestamp, pred) == ds$rating)
    }
  }

  data.frame('Dataset' = datasets,
             'Mode' = modes,
             'RMSE' = rmses,
             'Accuracy' = accuracies)
}
```

```r
knitr::kable(get_performance_metrics(ModeModel),
             caption = "Results for the Mode based model")
```

Table 1: Results for the Mode based model

Dataset	Mode	RMSE	Accuracy
Training	WHOLE	1.167044	0.2876016
Validation	WHOLE	1.168016	0.2874203

Dataset	Mode	RMSE	Accuracy
Training	PARTITIONED	1.167044	0.2876016
Validation	PARTITIONED	1.168016	0.2874203

Methods

Simple Average

```r
#' This object-constructor function is used to generate a model
#' that always returns as prediction the average of the rating in the
#' given dataset used to fit the model.
#' @param dataset The dataset used to fit the model
#' @return The model
AvgModel <- function(dataset) {
  model <- list()

  # The average of ratings
  model$mu <- mean(dataset$rating)

  #' The prediction function
  #' @param s The dataset used to perform the prediction of
  #' @return A vector containing the prediction
  model$predict <- function(s) {
    model$mu
  }

  model
}
```

Table 2: Results for the Average based model

Dataset	Mode	RMSE	Accuracy
Training	WHOLE	1.060331	0.2876016
Validation	WHOLE	1.061202	0.2874203
Training	PARTITIONED	1.059362	0.2876016
Validation	PARTITIONED	1.060221	0.2874203

Pseudo Linear Model

See: https://rafalab.github.io/dsbook/recommendation-systems.html