MovieLens

Dana Saeed

4 April 2021

Introduction

Recommendation systems plays an important role in e-commerce and online streaming services, such as Netflix, YouTube and Amazon. Making the right recommendation for the next product, music or movie increases user retention and satisfaction, leading to sales and profit growth. Companies competing for customer loyalty invest on systems that capture and analyses the user's preferences, and offer products or services with higher likelihood of purchase.

The economic impact of such company-customer relationship is clear: Amazon is the largest online retail company by sales and part of its success comes from the recommendation system and marketing based on user preferences. In 2006 Netflix offered a one million dollar prize2 for the person or group that could improve their recommendation system by at least 10%.

Usually recommendation systems are based on a rating scale from 1 to 5 grades or stars, with 1 indicating lowest satisfaction and 5 is the highest satisfaction. Other indicators can also be used, such as comments posted on previously used items; video, music or link shared with friends; percentage of movie watched or music listened; web pages visited and time spent on each page; product category; and any other interaction with the company's web site or application can be used as a predictor.

WorkFlow

The main steps in a data science project include:

Data preparation: download, parse, import and prepare the data to be processed and analysed. Data exploration and visualization: explore data to understand the features and the relationship between the features and predictors. Data cleaning: eventually the dataset contains unnecessary information that needs to be removed. Data analysis and modeling: create the model using the insights gained during exploration. Also test and validate the model. Communicate: create the report and publish the results. First we download the dataset from MovieLens website and split into two subsets used for training and validation. The training subset is called edx and the validation subset is called validation. The edx set is split again into two subsets used for training and and testing. When the model reaches the RMSE target in the testing set, we train the edx set with the model and use the validation set for final validation. We pretend the validation set is new data with unknown outcomes.

In the next step we create charts, tables and statistics summary to understand how the features can impact the outcome. The information and insights obtained during exploration will help to build the machine learning model.

Creating a recommendation system involves the identification of the most important features that helps to predict the rating any given user will give to any movie. We start building a very simple model, which is just the mean of the observed values. Then, the user and movie effects are included in the linear model, improving the RMSE. Finally, the user and movie effects receive regularization parameter that penalizes samples with few ratings.

Getting the data ready

First the dataset is downloaded and the required libraries are fetched.

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")

# 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")
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[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")
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")

# Add rows removed from 'validation' set back into 'edx' set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)

set.seed(123, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]
```

```
# Make sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

The edx set is used for training and testing, and the validation set is used for final validation to get the model prediction on unseen data. Here, we split the edx set in 2 parts: * the training set used to train the model (90 percent of total data) * the test set use to test the model (10 percent of the total data) * validation set , used to calculate the final RMSE.

Explorary Data Analysis

We need to understand the dataset and get few insights from it before moving forward. The str() method is used to get the idea about the datatypes of the data and to see the few of the values. We also check the dimensions of data to get the row and col count

```
head(edx)
```

```
##    userId movieId rating timestamp title genres
## 1:      1     122      5 838985046  <NA>   <NA>
## 2:      1     185      5 838983525  <NA>   <NA>
## 3:      1     292      5 838983421  <NA>   <NA>
## 4:      1     316      5 838983392  <NA>   <NA>
## 5:      1     329      5 838983392  <NA>   <NA>
## 6:      1     355      5 838984474  <NA>   <NA>
```

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  NA NA NA NA ...
##  $ genres   : chr  NA NA NA NA ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
dim(edx)
```

```
## [1] 9000055       6
```

Next we get the top five movie genres in the dataset

```
tp<- edx %>% group_by(genres) %>%
  summarise(n=n())
head(tp)
```

```
## # A tibble: 1 x 2
##   genres       n
##   <chr>    <int>
## 1 <NA>   9000055
```

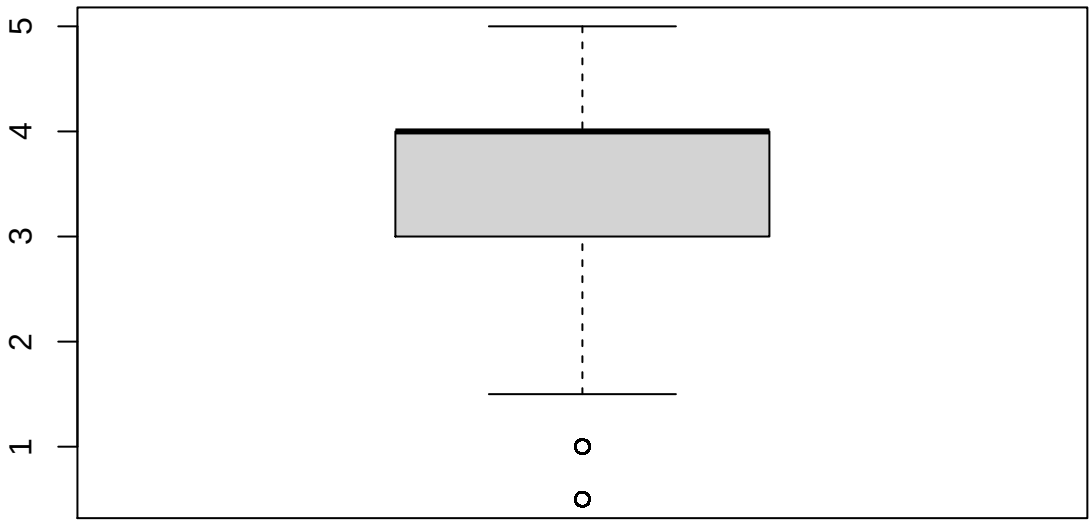
We see the ratings the users have given to the movies

```
tp<- edx %>% group_by(rating) %>% summarize(n=n())
head(tp)
```

```
## # A tibble: 6 x 2
##   rating       n
##    <dbl>   <int>
## 1    0.5   85374
## 2    1    345679
## 3    1.5  106426
## 4    2    711422
## 5    2.5  333010
## 6    3   2121240
```

Next the data is grouped by movie ID and we get the summary to see the distribution of the data and also see the boxplot of the data

```
boxplot(edx$rating)
```



Data Preparation

We only select those features that would give us the best performance without over complexity of the model.

```
train_set <- train_set %>% select(userId, movieId, rating, title)
test_set  <- test_set  %>% select(userId, movieId, rating, title)
```

Models to predict

The approach is to use different algorithms to make the predictions and calculate RMSE to get the best performing model, the best model is the one having the lowest error.

Predictions using Random probabilities

We use monte Carlo simulation to estimate the probability

```
set.seed(123, sample.kind = "Rounding")
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
# Create the probability of each rating
p <- function(x, y) mean(y == x)
rating <- seq(0.5,5,0.5)

# Estimate the probability of each rating with Monte Carlo simulation
B <- 10^3
M <- replicate(B, {
  s <- sample(train_set$rating, 100, replace = TRUE)
  sapply(rating, p, y= s)
})
prob <- sapply(1:nrow(M), function(x) mean(M[x,]))

# Predict random ratings
y_hat_random <- sample(rating, size = nrow(test_set),
                       replace = TRUE, prob = prob)


mse <- function(yhat,y)
{
return(  mean((yhat - y)^2))
}
# Create a table with the error results
pred <- tibble(Method = "Project Goal", RMSE = 0.8649, MSE = NA, MAE = NA)
pred <- bind_rows(pred,
                  tibble(Method = "Monte Carlo Random prediction",
                         RMSE = caret::RMSE(test_set$rating,y_hat_random),
                         MSE  = mse(test_set$rating, y_hat_random),
                         MAE  = caret::MAE(test_set$rating,y_hat_random)))


pred
```

```
## # A tibble: 2 x 4
##   Method                          RMSE   MSE   MAE
##   <chr>                          <dbl> <dbl> <dbl>
## 1 Project Goal                   0.865 NA    NA
## 2 Monte Carlo Random prediction 1.50   2.25  1.16
```

#Linear Model for predictions

```r
# Mean of observed values
mu <- mean(train_set$rating)

# Update the error table
pred <- bind_rows(pred,
                  tibble(Method = "Mean model",
                         RMSE = caret::RMSE(test_set$rating, mu),
                         MSE  = mse(test_set$rating, mu),
                         MAE  = caret::MAE(test_set$rating, mu)))
# Show the RMSE improvement
pred
```

```
## # A tibble: 3 x 4
```

```
##   Method                           RMSE   MSE    MAE
##   <chr>                           <dbl> <dbl>  <dbl>
## 1 Project Goal                    0.865 NA    NA
## 2 Monte Carlo Random prediction 1.50   2.25  1.16
## 3 Mean model                      1.06   1.12  0.856
```

After getting the predictions on the linear model, the next step is to include the effect of the movie on the linear model so the extension to the linear model can be defined from the code below

```
# Movie effects (bi)
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
head(bi)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.414
## 2       2 -0.314
## 3       3 -0.363
## 4       4 -0.648
## 5       5 -0.442
## 6       6  0.299
```

```
#predict

y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i

# Calculate the RMSE
pred <- bind_rows(pred,
                  tibble(Method = "Mean + bi",
                         RMSE = caret::RMSE(test_set$rating, y_hat_bi),
                         MSE  = mse(test_set$rating, y_hat_bi),
                         MAE  = caret::MAE(test_set$rating, y_hat_bi)))

# Show the RMSE improvement
pred
```

```
## # A tibble: 4 x 4
##   Method                           RMSE    MSE    MAE
##   <chr>                           <dbl>  <dbl>  <dbl>
## 1 Project Goal                    0.865 NA     NA
## 2 Monte Carlo Random prediction 1.50   2.25   1.16
## 3 Mean model                      1.06   1.12   0.856
## 4 Mean + bi                       0.943  0.890  0.738
```

In the step by step approach the model is now added with the user affect to see the RMSE of the model on the test set

```
# User effect (bu)
bu <- train_set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Prediction
y_hat_bi_bu <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Update the results table
pred <- bind_rows(pred,
                  tibble(Method = "Mean + bi + bu",
                         RMSE = caret::RMSE(test_set$rating, y_hat_bi_bu),
                         MSE  = mse(test_set$rating, y_hat_bi_bu),
                         MAE  = caret::MAE(test_set$rating, y_hat_bi_bu)))

# Show the RMSE improvement
pred
```

```
## # A tibble: 5 x 4
##   Method                          RMSE    MSE    MAE
##   <chr>                          <dbl>  <dbl>  <dbl>
## 1 Project Goal                   0.865 NA     NA
## 2 Monte Carlo Random prediction 1.50   2.25   1.16
## 3 Mean model                     1.06   1.12   0.856
## 4 Mean + bi                      0.943  0.890  0.738
## 5 Mean + bi + bu                 0.865  0.748  0.669
```

The RMSE improved from the initial estimation based on the mean. However, we still need to check if the model makes good ratings predictions. Check the 15 largest residual differences. We also look at the table for the top best and top worst movies.

```
train_set %>%
  left_join(bi, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:15)
```

```
##     userId movieId rating title        b_i  residual
##  1:  26423    6483    5.0  <NA> -2.6135840  4.101124
##  2:  29924    6483    5.0  <NA> -2.6135840  4.101124
##  3:   2507     318    0.5  <NA>  0.9410261 -3.953487
##  4:   7708     318    0.5  <NA>  0.9410261 -3.953487
##  5:   9214     318    0.5  <NA>  0.9410261 -3.953487
##  6:   9568     318    0.5  <NA>  0.9410261 -3.953487
##  7:   9975     318    0.5  <NA>  0.9410261 -3.953487
##  8:  10680     318    0.5  <NA>  0.9410261 -3.953487
##  9:  10749     318    0.5  <NA>  0.9410261 -3.953487
```

```
## 10:  13496     318    0.5  <NA>  0.9410261 -3.953487
## 11:  21710     318    0.5  <NA>  0.9410261 -3.953487
## 12:  24312     318    0.5  <NA>  0.9410261 -3.953487
## 13:  25239     318    0.5  <NA>  0.9410261 -3.953487
## 14:  26260     318    0.5  <NA>  0.9410261 -3.953487
## 15:  26312     318    0.5  <NA>  0.9410261 -3.953487
```

```
titles <- train_set %>%
  select(movieId, title) %>%
  distinct()

bi %>%
  inner_join(titles, by = "movieId") %>%
  arrange(-b_i) %>%
  select(title) %>%
  head()
```

```
## # A tibble: 6 x 1
##   title
##   <chr>
## 1 <NA>
## 2 <NA>
## 3 <NA>
## 4 <NA>
## 5 <NA>
## 6 <NA>
```

```
bi %>%
  inner_join(titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title) %>%
  head()
```

```
## # A tibble: 6 x 1
##   title
##   <chr>
## 1 <NA>
## 2 <NA>
## 3 <NA>
## 4 <NA>
## 5 <NA>
## 6 <NA>
```

Matrix Factorizations

Matrix factorization approximates a large user-movie matrix into the product of two smaller dimension matrices. Information in the train set is stored in tidy format, with one observation per row, so it needs to be converted to the user-movie matrix before using matrix factorization. This code executes this transformation.

```
train_data <- train_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()

if(!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: recosystem
```

```
## Warning: package 'recosystem' was built under R version 4.0.5
```

```
set.seed(123, sample.kind = "Rounding") # This is a randomized algorithm
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# Convert the train and test sets into recosystem input format
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
# Create the model object
r <-  recosystem::Reco()
# Select the best tuning parameters
opts <- r$tune(train_data, opts = list(
                                       lrate = c(0.1, 0.2),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = 4, niter = 1))
 # Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 1))
```

```
## iter      tr_rmse          obj
##    0       0.9708   8.3107e+06
```

```
y_hat_reco <-  r$predict(test_data, out_memory())
head(y_hat_reco, 10)
```

```
##  [1] 4.090288 4.215611 2.480450 3.795496 3.335462 3.246420 3.037222 4.180341
##  [9] 3.419130 4.139817
```

```
pred<- bind_rows(pred,
                 tibble(Method = "Matrix Factorization",
                        RMSE = caret::RMSE(test_set$rating, y_hat_reco),
                        MSE  = mse(test_set$rating, y_hat_reco),
                        MAE  = caret::MAE(test_set$rating, y_hat_reco)))
pred
```

```
## # A tibble: 6 x 4
##   Method                           RMSE    MSE    MAE
##   <chr>                           <dbl>  <dbl>  <dbl>
## 1 Project Goal                   0.865 NA     NA
## 2 Monte Carlo Random prediction 1.50   2.25   1.16
## 3 Mean model                     1.06   1.12   0.856
## 4 Mean + bi                      0.943  0.890  0.738
## 5 Mean + bi + bu                 0.865  0.748  0.669
## 6 Matrix Factorization           0.881  0.777  0.683
```

As we have seen from RMSE Matrix Factorization performed better than other models and we were supposed to do validation on the best model so in the next step we validate the model . #Validation on Matrix Factorization

```
set.seed(1234, sample.kind = "Rounding")
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# Convert 'edx' and 'validation' sets to recosystem input format
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))
validation_reco  <-  with(validation, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))
# Create the model object
r <-  recosystem::Reco()
# Tune the parameters
opts <-  r$tune(edx_reco, opts = list(
                                      lrate = c(0.1, 0.2),

                                      costq_l2 = c(0.01, 0.1),
                                      nthread  = 4, niter = 1))
# Train the model
r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 1))
```

```
## iter      tr_rmse          obj
##    0       0.9582   8.9485e+06
```

```
# Calculate the prediction
y_hat_final_reco <-  r$predict(validation_reco, out_memory())

# Update the result table
pred <- bind_rows(pred,
                  tibble(Method = "Final Matrix Factorization ",
                         RMSE = caret::RMSE(validation$rating, y_hat_final_reco),
                         MSE  = mse(validation$rating, y_hat_final_reco),
                         MAE  = caret::MAE(validation$rating, y_hat_final_reco)))
```

Conclusion

The first step was to start with the simple linear model and then adding more and complexity by adding the effects of the movies and the users. The RMSE was satisfying and reduced with the model complexity indicating that more and more complex model would have the better predictions so we decided to use Matrix Factorization to create the model and found out that it's performance was even better than others so as equired the hold out validation is performed on it and we produced some good results