MovieLens Report

Rodrigo Caballero

20 de noviembre de 2019

Executive Summary

This project works on the movieLens 10M dataset, looks to create a prediction algorithm on the rating variable. The goal of the algorithm is to reduce RMSE of the prediction in a validation set, that accounts 10% of the whole dataset.

Dataset

MovieLens dataset consist on a 10M rows and 6 variables, 5 features and a target "rating".

Features

1. userId: an integer, each of a different user in the platform
2. movieId: an integer, each of a different movie in the platform
3. timestamp: an integer, given the time the rating was made
4. title: a character, for movie title
5. genres: a character, for the movie genre.

Target

As the goal of the algorithm is to reduce RMSE below 0.8649, target variable would be treated as a "double" variable.

- Rating: a double, rating given by the user

Key Steps to be performed

The following steps were required for the project

1. Loading libraries and setting up sets
2. Data exploratory analysis with key insights
3. Preprocessing and Train algorithm in low scale
4. RMSE results in low scale
5. Preprocesing and training final algorithm
6. Prediction results (RMSE) on Validation dataset

Set up

Libraries

Following libraries will be used

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------- tidyverse 1.2.1 --

## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ---------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.6.1

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.1

## Loading required package: lattice

##
## Attaching package: 'caret'

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

```
library(anytime)
```

```
## Warning: package 'anytime' was built under R version 3.6.1
```

```
library(lubridate)
```

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

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year

## The following object is masked from 'package:base':
##
##     date
```

```
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(class)
```

Create edx and validation set

Code followed for the creation of the sets was the given by the project. The RScript explains it in detail. Average time for this process in a small laptop is 15 minutes.

Exploratory analysis

Data partition

To make code efficient, exploratory analysis will be a random sample of 20K observations of the edx dataset.

```
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
data_exp_index <- createDataPartition(y = edx$rating, times = 1, p = 0.002222,
                                      list = FALSE)
data_exp <- edx[data_exp_index,]
```

Data wrangling

Some movies has several genres associated, but several does not, hence the first genre of the movie will be accounted in the algorithm so we don't generate NA's in the dataset. The rest of the genres would be "dropped"

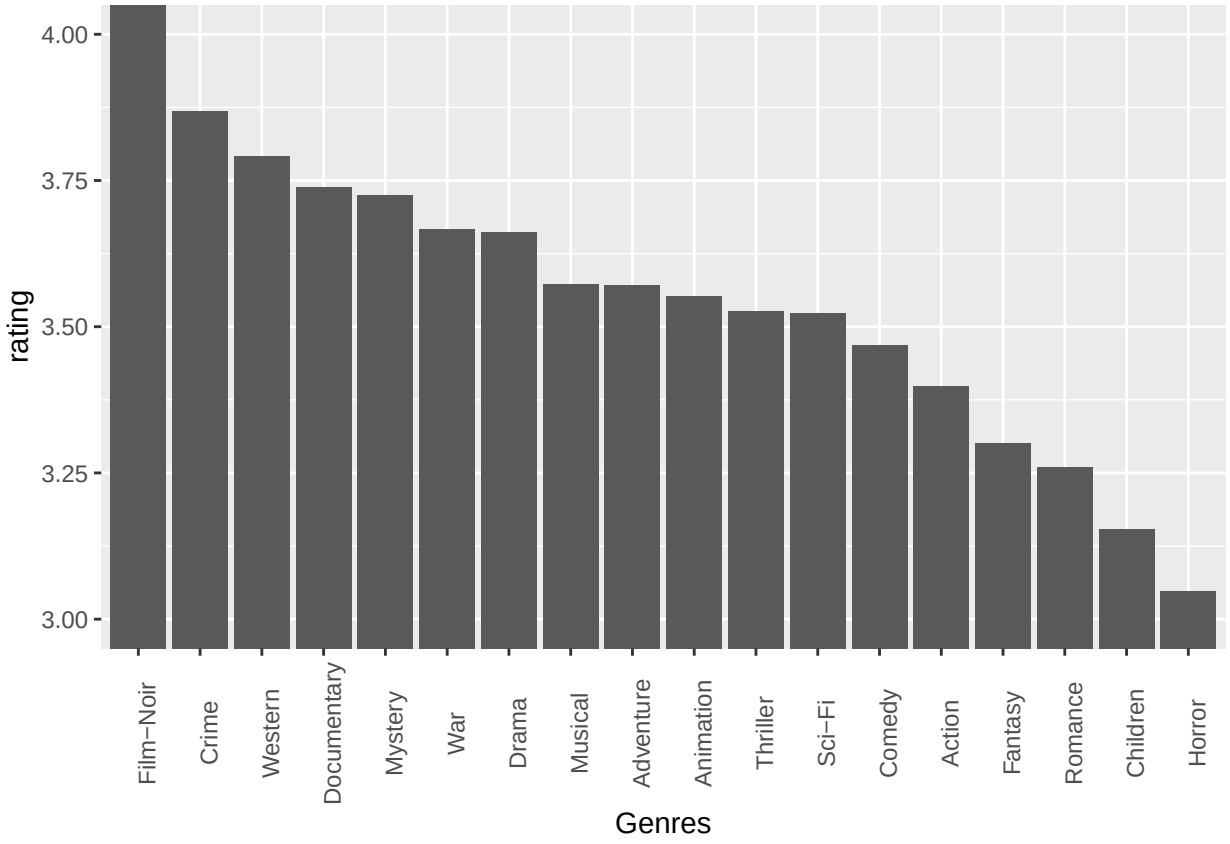
```
data_exp <- data_exp %>%
  separate(genres, sep = '[|]',extra = 'drop',
           into = 'genres')
```

Rating by genre

Some genres are better rated on average than others

```
genre_feat <- data_exp %>%
  group_by(genres) %>%
  summarize(rating = mean(rating)) %>%
  arrange(desc(rating))

genre_feat %>%
  ggplot(aes(x = reorder(genres, -rating), rating)) +
  geom_bar(stat = 'Identity') +
  xlab('Genres')+
  coord_cartesian(ylim = c(3,4)) +
  theme(axis.text.x = element_text(angle = 90))
```



Rating by number of ratings per movie

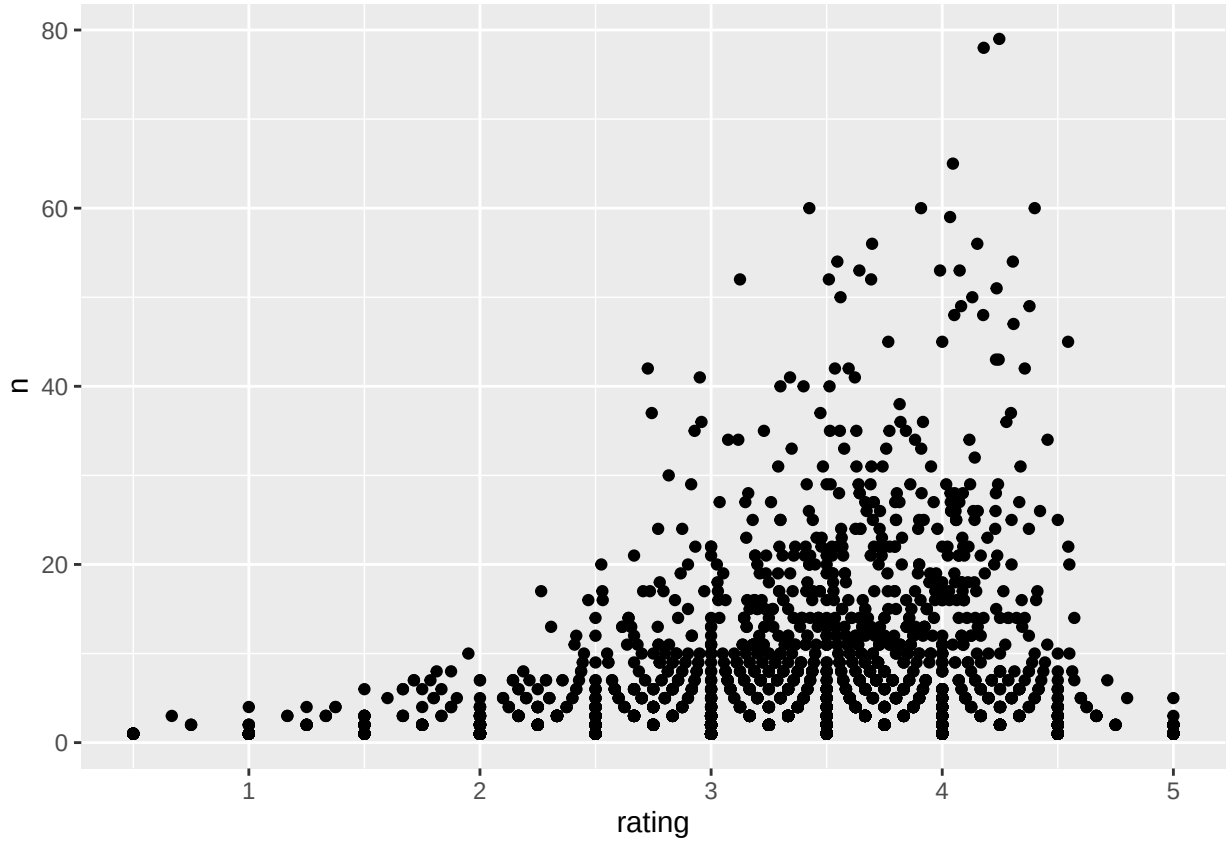
The number of ratings a movie has on the platform has a normal distribution effect on the average rating of the movie. For example, a well-know movie like Toy Story or Jurassic Park that has more ratings, the

average of its ratings will tend to the overall mean (3.5 for edx dataset), and movies with little ratings will tend to move out the average.

The average rating of the movie is an important factor, independent of the user or the genre, this will show movies that are overall well received by the users on the platform. We can see this effect visually in the next graph, in the case of the movie close to 80 ratings and above 4 on average ratings.

```
n_feat <- data_exp %>%
  group_by(title) %>%
  summarize(n = n(), rating = mean(rating)) %>%
  arrange(desc(n))

n_feat %>%
  ggplot(aes(rating, n)) +
  geom_point()
```

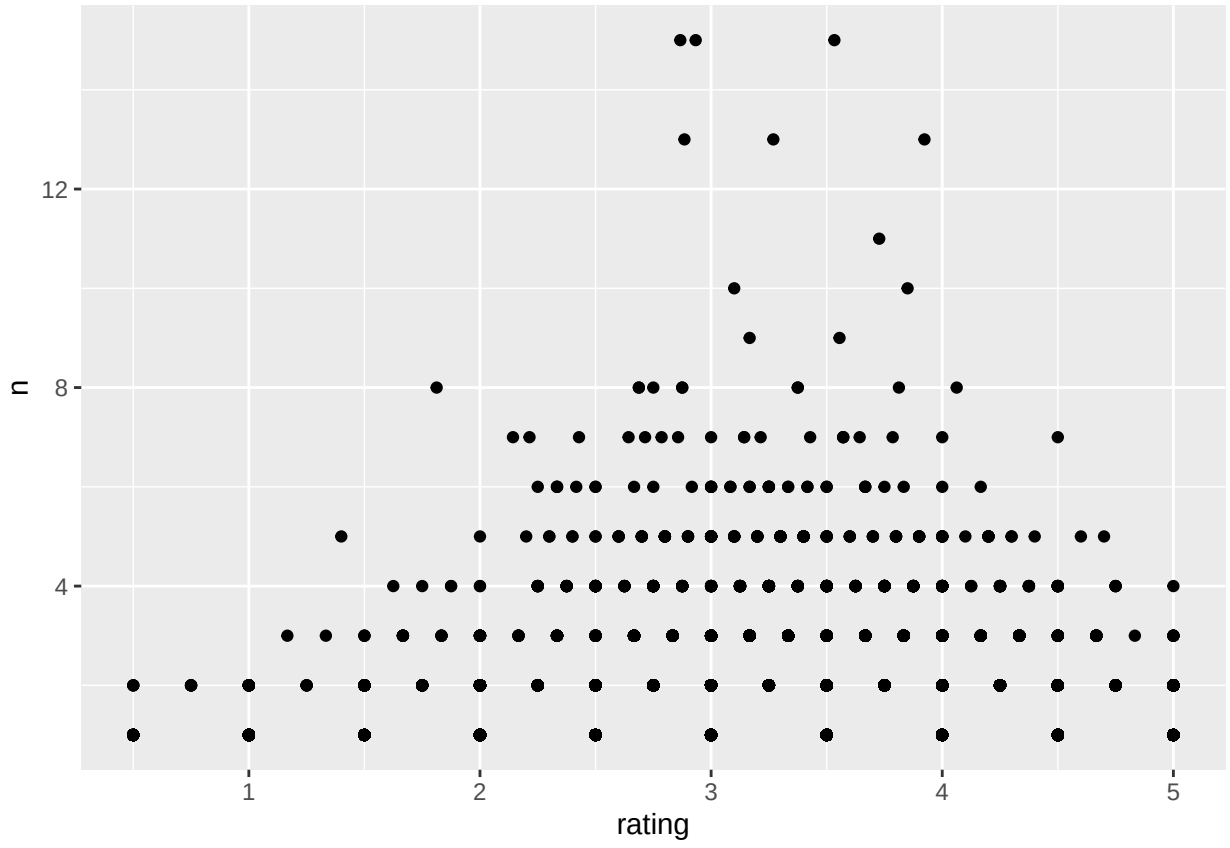


Rating by number of ratings per user

We see the same effect as the previous example, the more active a user is in the platform, the average of it's ratings will fall in the overall mean.

```
user_feat <- data_exp %>%
  group_by(userId) %>%
  summarize(n = n(), rating = mean(rating)) %>%
  arrange(desc(n))
```

```
user_feat %>%
  ggplot(aes(rating, n)) +
  geom_point()
```



Timestamp factor

A timestamp analysis was intended by creating a timeseries and create weights for the prediction given a cyclical, seasonal or random trend. Nevertheless due to the fact that timestamp is multivariate this analysis became complex to execute.

Insights of the exploratory analysis

These are the key insights that results from the previous exploratory analysis

- Some genres are better rated than others, we will also include on the algorithm the preference of certain users to certain genres by the average of rating per genre per user.
- Some movies are well received on the platform given its overall rating, independent of the genre.
- The number of ratings per movie and user has a normal distribution effect that we can include in the algorithm.
- Timeseries analysis was not viable at the moment given the multivariate nature of the timestamp factor.

Preprocesing and train algorithm in low scale

The objective of this step is to see behavior of an algorithm with the insights of exploratory analysis in the low scale to take the algorithm to whole validation dataset.

Adding new variables

The following variables will be added for the prediction:

- NUserRatings = an integer, number of ratings the user has in the platform
- NMovieRatings = an integer, number of ratings a movie has in the platform
- avg_rating_movie = a double, the average rating a movie has in the platform
- avg_rating_genre = a double, average rating a genre has in the platform
- avg_rating_user_genre = a double, the average rating a user has per genre

```
data_exp <- data_exp %>%
  group_by(userId) %>%
  mutate(NUserRatings = n()) %>%
  ungroup()

data_exp <- data_exp %>%
  group_by(movieId) %>%
  mutate(NMovieRatings = n()) %>%
  ungroup()

data_exp <- data_exp %>%
  group_by(movieId) %>%
  mutate(avg_rating_movie = mean(rating)) %>%
  ungroup()

data_exp <- data_exp %>%
  group_by(genres) %>%
  mutate(avg_rating_genre = mean(rating)) %>%
  ungroup()

data_exp <- data_exp %>%
  group_by(userId, genres) %>%
  mutate(avg_rating_user_genre = mean(rating)) %>%
  ungroup()
```

Preprocessing

Mutate and select variables

As well, we will transform the genre variable as a factor, will exclude the rest of the variables

```
data_exp <- data_exp %>%
  mutate(genres_factor = as.integer(as.factor(genres)))

data_exp <- data_exp %>%
  select(rating, NUserRatings, NMovieRatings, avg_rating_movie,avg_rating_genre, avg_rating_user_genre)
```

Extract y vector, normalize x

We will extract the target and transform it into a vector, as well we will normalize the predictors and create a training set and test set (10%) of the data exploration set of 20K observations, this as a cross-validation strategy.

```
#We are extracting the y vector
y_exp <- data_exp['rating']
x_exp <- data_exp[,2:6]

#We normalize the x vectors
x_exp <- as.data.frame(scale(x_exp))


set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(y = x_exp$NUserRatings,times = 1,p = 0.1,list = FALSE)

#SETS FOR X
train_set_exp <- x_exp[-test_index,]
test_set_exp <- x_exp[test_index,]


#VECTOR OF Y
y_train_exp <- y_exp[-test_index,,drop =TRUE]
y_test_exp <- y_exp[test_index,,drop = TRUE]
```

Training algorithm in low scale

Now we are training a KNN Algorithm to make predictions and calculate RMSE in the low scale. Average time of this process in a small laptop is less than 5 minutes.

```
train_knn <- train(train_set_exp,y = y_train_exp,method = 'knn')

y_hat_knn <- predict(train_knn, test_set_exp)


y_results <- as.data.frame(y_test_exp)
y_results['predicted'] <- as.data.frame(y_hat_knn)

y_results <- y_results %>%
  mutate(RMSE = ((predicted - y_test_exp)^2)/n())

RMSE_knn9 = sqrt(sum(y_results$RMSE))

print(RMSE_knn9)
```

```
## [1] 0.2664538
```

We have very good result of the algorithm with an RMSE of around 0.26, the best tunning parameter for the number of neighbors is K = 9

To remember the RMSE function is the following

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Preprocesing and training algorithm for Validation dataset

As we saw good results in the low scale we will take the algorithm to predict the whole validation dataset and calculate RMSE. Steps of preprocesing would be similar to the ones taken in the low scale.

We will fit a knn algorithm with a tunning parameter of k = 9.

Preprocessing of edx dataset

Dataset would have same preprocesing as in low scale, nevertheless, for computing reasons, training of the algorithm would not be able to run on the whole dataset, so Data Partition in a random sample 3 times larger than data exploration set would be performed.

```
edx <- edx %>%
  separate(col = genres, into = 'genres',sep = '[|]',
           extra = 'drop')

validation <- validation %>%
  separate(col = genres, into = 'genres', sep = '[|]',
           extra = 'drop')

edx <- edx %>%
  group_by(userId) %>%
  mutate(NUserRatings = n()) %>%
  ungroup()

edx <- edx %>%
  group_by(movieId) %>%
  mutate(NMovieRatings = n()) %>%
  ungroup()

edx <- edx %>%
  mutate(genres_factor = as.integer(as.factor(genres)))

edx <- edx %>%
  group_by(movieId) %>%
  mutate(avg_rating_movie = mean(rating)) %>%
  ungroup()

edx <- edx %>%
  group_by(genres) %>%
  mutate(avg_rating_genre = mean(rating)) %>%
  ungroup()
```

```
edx <- edx %>%
  group_by(userId, genres) %>%
  mutate(avg_rating_user_genre = mean(rating)) %>%
  ungroup()

edx <- edx %>%
  select(rating, NUserRatings, NMovieRatings, avg_rating_movie,avg_rating_genre, avg_rating_user_genre)

set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
train_index <- createDataPartition(y = edx$rating, times = 1, p = 0.002222*3,
                                   list = FALSE)
edx1 <- edx[train_index,]
```

Preprocesing of validation dataset

Same preprocesing goes to validation dataset

```
validation <- validation %>%
  group_by(userId) %>%
  mutate(NUserRatings = n()) %>%
  ungroup()

validation <- validation %>%
  group_by(movieId) %>%
  mutate(NMovieRatings = n()) %>%
  ungroup()

validation <- validation %>%
  mutate(genres_factor = as.integer(as.factor(genres)))

validation <- validation %>%
  group_by(movieId) %>%
  mutate(avg_rating_movie = mean(rating)) %>%
  ungroup()

validation <- validation %>%
  group_by(genres) %>%
  mutate(avg_rating_genre = mean(rating)) %>%
  ungroup()

validation <- validation %>%
  group_by(userId, genres) %>%
  mutate(avg_rating_user_genre = mean(rating)) %>%
  ungroup()

validation <- validation %>%
  select(rating, NUserRatings, NMovieRatings, avg_rating_movie,avg_rating_genre, avg_rating_user_genre)
```

Extract y vector, normalize x

As the same case we did for the low scale datasets, we will extract the target in a numeric vector and normalize predictors

```
#We will get the vectors for the outcome (y) for the edx
#and validation dataset

y_edx <- edx1['rating']
y_validation <- validation['rating']

#We transform outcome in a vector

y_edx <- y_edx[,,drop = TRUE]
y_validation <- y_validation[,,drop = TRUE]

#We also are getting the matrix of the predictors to do the
#preprocesing

x_edx <- edx1[,2:6]
x_validation <- validation[,2:6]

#Now we do preprocesing of the matrix in the edx and validation
#dataset

x_edx <- as.data.frame(scale(x_edx))
x_validation <- as.data.frame(scale(x_validation))
```

Training of algorithm

As we mentioned before, we will train a knn with 9 neighbors with the sample of edx dataset. This process would take around 10 minutes on a small laptop

```
knn_grid <- expand.grid(k = 9)

train_knn <- train(x = x_edx, y = y_edx, method = 'knn', tuneGrid = knn_grid)
```

Prediction on Validation dataset and RMSE results

For the final step,we would make prediction on the validation dataset and review the result of the RMSE. This process would take on average 15 minutes on a small laptop.

```
y_hat_knn <- predict(train_knn, x_validation)


y_results <- as.data.frame(y_validation)
y_results['predicted'] <- as.data.frame(y_hat_knn)

y_results <- y_results %>%
  mutate(RMSE = ((predicted - y_validation)^2)/n())
```

```r
RMSE_knn9 = sqrt(sum(y_results$RMSE))

print(RMSE_knn9)
```

```
## [1] 0.7895488
```

Resulting RMSE on validation dataset is around 0.78, as RMSE is below the target RMSE of 0.8649, the algorithm results successful.

Conclusions

The algorithm trained, a knn with 9 neighbors, used for the prediction of the validation dataset gives desired results.

Limitations and Future Work

Some limitations, that if they were overcome it would improve the effectiveness of the algorithm, include:

- Computing limitations, that doesn't allow us to train the algorithm in a larger dataset than the train set (validation set) and that gives us a higher RMSE in the Validation than in the low scale, as well as comparing the effectiveness of several algorithms.

Future work to overcome this limitation is computing power, Big Data / Distibutive computing scheme workflow, like Python library Spark, or a more robust cross-validation strategy

- Timestamp factor, as a multivariate variable, can not allow us to made a timeseries traditional analysis and get weights given cyclical, seasonal or random trends, for example a Holt-Winters model with univariate variables.

Future work to overcome this limitation is implementatioN of a model that allow multivariate analysis for time series.