Juan Carlos Coloma López – coloma@alum.mit.edu

Introduction and executive summary

The following report presents the prediction done on movielens database. It is composed of around 10m entries, with a limited amount of columns (movie ID number, customer ID number, review date, title plus release year, genre). It does present more data than columns as, for instance in the title attribute we can find also the year. In addition, column genre presents, for many movies, several entries.

The database is divided into two major chunks: training set (named as edx) plus the validation set (named this way). Validation set accounts for 10% while edx for the rest.

The analysis run, first of all aims to improve the quality of data (converting the genres column into a numeric one, plus further analysis explained in the code section). Afterwards several data mining analysis are run in the edx set; and then validated into the validation set. Overall the target is to predict the movie rating. Obviously this exercise must be done at individual movie and individual customer level.

The main difficulty that this database presents is the size. Very often, during the code construction I have faced errors of memory consumption, of lack of available data management capacity or just "fatal error". There are many actions presented in the appendices that I would have liked to run, but I have not been able due to these limitations.

As a consequence, results are not close to an optimal figure. The best RMSE presented achieves 0.957.

Methods and analysis section

Data exploration and visualization

An initial analysis run with simple and straight functions such as head() or str() reveals a reduced database in terms of attributes (it has only 6 columns) with a lot of information packed into it; and an enormous database in terms of entries (close to 10m).

The structure of the database is as follows. Several columns compose it:

```
str(edx)
'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 83898
3653 838984885 838983707 838984596 ...
 $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate
(1994)" ...
 $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thr
iller" "Action|Adventure|Sci-Fi" ...
```

As we can see, we have several columns not formatted appropriately: timestamp (should be formatted as a fate, while R reads it as an integer, and genres, which is clearly a categorical attribute and is stored as a character).

I have tried several techniques to improve data quality but one of them has been disregarded as it increased dramatically the size of the data frames, generating capacity constraint errors and preventing data mining methods from running.

The following table (Table 1) presents a list of the techniques used to improve data quality and the final status. The code of each of them is presented into the appendices.

Technique	Description and target	Status
Genre Identification	Individualization of genres and link to a numeric value.	Used
Title extraction	Using regex function, extraction of title and year.	Used
Transformation of timestamp into a time format	Extraction of year of movie review	Used
Genres wrangling	Separation of genres into individual lines and generation of dummy variables	Discarded

Table 1: list of data wrangling techniques

As stated above, I have tried all these techniques. They definitively produce a wrangled database. Hence, from a theoretical standpoint they should be performed. The blocker to go ahead with the last one, *Genres wrangling*, is data consumption. All of them, at least in my laptop, produce subsequent capacity errors. To overcome this situation I did a workaround: selecting the unique values among the genres category and assign them a numeric value to make an approximation to a cleaner categorical attribute.

For data analysis and structure I have run a principal component analysis (the code is shown in the later coding section) selecting only numeric variables (therefore eliminating timestamp, title and title without the year, and genres columns). The result shown is pretty interesting.

First, using summary(edxpca) we obtain:

```
Importance of components:
                            PC1        PC2       PC3   PC4   PC5   PC6
Standard deviation     2.059e+04 8941.9918 107.84936 13.06 3.438 1.051
Proportion of Variance 8.412e-01    0.1587   0.00002  0.00 0.000 0.000
Cumulative Proportion  8.412e-01    1.0000   1.00000  1.00 1.000 1.000
```

This involves that just with PC1 we hold more than 80% of total variability, therefore PC1 provides a sufficiently robust approximation.

Second, using edxpca$rotation we can obtain:

```
edxpca$rotation
                PC1           PC2           PC3           PC4           PC5           PC6
userId    9.999965e-01  2.641849e-03  7.933462e-06  6.785270e-07  2.770754e-06 -1.464678e-07
movieId   2.641828e-03 -9.999945e-01  1.953372e-03 -3.987245e-04  1.486564e-04 -5.279621e-06
rating    1.187866e-07  7.767197e-07  1.330344e-04 -1.029122e-02  1.138312e-02  9.998822e-01
year      1.420650e-07 -3.646043e-04  1.725007e-02  9.997949e-01  2.680403e-03  1.025751e-02
year_rev  3.193720e-06 -1.538626e-04 -2.112896e-03  2.600093e-03 -9.999293e-01  1.141070e-02
genresId  1.309163e-05 -1.959604e-03 -9.998471e-01  1.724153e-02  2.161119e-03  2.858853e-04
```

Therefore, userId is very representative in terms of approximation to the ortogonal vector PC1

As a second method for data exploration I chose the correlation matrix (Table 2):

	userId	movieId	rating	year	year_rev	genresId
userId	1.00	0.00	0.00	0.00	0.02	0.00
movieId	0.00	1.00	-0.01	0.24	0.37	0.16
rating	0.00	-0.01	1.00	-0.12	-0.04	-0.01
year	0.00	0.24	-0.12	1.00	0.09	-0.09
year_rev	0.02	0.37	-0.04	0.09	1.00	0.12
genresId	0.00	0.16	-0.01	-0.09	0.12	1.00

Table 2 Correlation matrix

We can see that the maximum correlation index belongs to the pair year_rev ~ movieId. This makes sense from a logic standpoint, as users will tend to review recent movies (hot movies we could say). However, from a mathematical perspective we cannot eliminate any variable. The second consideration is with regards to the 'Rating' variable. First, there is no significant correlation among the variable we are trying to predict (rating) and any of the rest. Second, the more correlated variables are #1year, and #2 year_rev, #3genresId and #4 movieId.

Methods.

The methods used have depended heavily on capacity constraints errors.

First method run is a linear regression. For this method, non-numeric variables have been removed. This is one of the most simple machine learning techniques, however, due to its simplicity is able to provide results, even with this heavy database.

The second technique used is Knn. My target was to run several iterations from k = 1 to k = n (big number). However, this technique presents severe blockers in terms of capacity constraints. The code provided shows the spirit of the exercise, but this code has not been able to provide results.

The third technique used is classification tree. This technique has been able to provide better results than the linear regression, however does not reach enough accuracy as measured by RMSE as expected.

The fourth method employed is random forest. This technique, like knn, has presented severe capacity constraint blockers. Therefore, unfortunately, has not been able to provide results.

Insights

Using the code summary(fit_lm) we obtain the following results:

```
> summary(fit_lm)

Call:
lm(formula = rating ~ ., data = edx_prc)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7933 -0.5356  0.1370  0.6030  1.8316

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept)  4.452e+01  2.104e-01  211.593   <2e-16 ***
userId       1.436e-07  1.702e-08    8.434   <2e-16 ***
movieId      5.095e-06  4.382e-08  116.286   <2e-16 ***
year        -1.019e-02  2.683e-05 -379.979   <2e-16 ***
year_rev    -1.035e-02  1.019e-04 -101.510   <2e-16 ***
genresId    -2.871e-04  3.289e-06  -87.289   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.051 on 9000049 degrees of freedom
Multiple R-squared:  0.01721,  Adjusted R-squared:  0.01721
F-statistic: 3.152e+04 on 5 and 9000049 DF,  p-value: < 2.2e-16
```

This means, first of all and based in R-sq adjusted, that this regression is not really significant for an accurate prediction. However, the coefficients tell us that:

- UserId has a positive effect in the rating (provided userId is a number assigned based in sign in date, the most tenured customers tend to provide lower ratings than new ones).
- The rest (movieId, year, year_rev, genresId) tend to have a negative effect on the rating. For the first three attributes this can mean that the newer the movie, the lower tends to be its rating.

Results

The following table (Table 3) presents the results of the discussed techniques.

Technique	RMSE
Linear Regression	1.052
Knn	N/A
Classification Tree	0.957
Random Forest	N/A

Table 3: list of techniques and result obtained (RMSE)

As discussed previously, best RMSE obtained is far from an expected result. This output has been heavily conditioned by the capacity constraint blockers found in R.

The best RMSE is driven by the Classification tree, 0.957.

Conclusion

This database presents a suggestive problem of rating prediction through data mining technique. The factor that most heavily constraints all the analysis that can be run is the massive data size. However, several techniques can be applied to improve prediction. In my case, the only method that presents results is the linear regression.

R code

The following code presents analysis run to train and test a model to determine movie ratings.

The first part of the code presents the load of needed libraries:

```
library(tidyverse)
library(lattice)
library(caret)
library(tidyr)
library(ModelMetrics)
library(purrr)
library(rpart)
library(randomForest)
library(recipes)
library(varhandle)
```

Then, we can find the code suggested in the problem statement:

```
setwd("~/2.Education/1.R/Harvardx/Captsone/ml-10m")
ratings <- read.table(text = gsub("::", "\t", readLines("ratings.dat")),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines("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)
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)
```

We have many different genres, and especially, that many movies present various genres. Hence, the following code presents a way to translate the text column of genre to a numeric one.

```
##wrangling data
```

Separating the title into year and title, then transforming timestamp into a date format, and last extracting the year from the timestamp.

```
edx <- edx %>% extract(title, c("title_noY", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F)

edx$timestamp <- as_datetime(edx$timestamp)

edx$year <- as.numeric(edx$year)

edx <- edx %>% mutate(year_rat = year(timestamp))


validation <- validation %>% extract(title, c("title_noY", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$",
remove = F)

validation$timestamp <- as_datetime(validation$timestamp)

validation$year <- as.numeric(validation$year)
```

```
validation <- validation %>% mutate(year_rat = year(timestamp))


### Adjusting the columns to numeric values

genr <- data.frame(unique(edx$genres))

vec <- c()

n <- as.numeric(nrow(genr))

for (i in 1:n){

  vec[i] <- i

  }

genr <- genr %>% mutate(genresId = vec)

names(genr) <- c("genres", "genresId")


edx <- edx %>% left_join(genr, by = "genres")

validation <- validation %>% left_join(genr, by = "genres")

remove(genr,i, n, vec)
```

The following piece of code runs a PCA. The output is that all the numeric variables are representative. The reason is that all of them explain, to some extent, the database's variability.

```
### running a PCA

ind <- c(1:3,6,7,9,10)

edx_prc <- edx[ind]

edxpca <- prcomp(edx_prc)

edxpca$rotation

### running a correlation matrix

ind <- c(1,2,6,7,9,10)

edx_cor <- edx[ind]

cor(edx_cor)
```

##

###First method: linnear regression

The following piece of code presents the first method used to predict rating, the linear regression.

```
names(edx)

str(edx)

ind <- c(-5,-6)

edx_prc <- edx[ind]

edxpca <- prcomp(edx_prc)

fit_lm <- lm(rating ~ ., data = edx_prc)

edx_hat_lm <- predict(fit_lm, newdata=validation)

rmse(validation$rating,edx_hat_lm)

##rmse_reported: 1.06
```

##

###Second method: knn

The second method used in Knn. Using an iteration to test several values of k. Disclaimer: the following piece of code provides an error of memory consumption. Even trying just one iteration, with several values of k: 100000, 1000000, etc. However, at least I wanted to present which is the spirit of the knn analysis.

```
saveRDS(edx, "edx_it1.rds")

saveRDS(validation, "valid_it1.rds")

saveRDS(fit_knn, "fit_knn_it1.rds")


edx <- readRDS("edx_it1.rds")

validation <- readRDS("valid_it1.rds")
```

```
ks <- seq(3, 900003, 100000)

accurac <- map_df(ks, function(k){

 fit_knn <- knn3(rating ~ userId+movieId+timestamp+genresId, data = edx, k = 100000)

 edx_hat_knn <- predict(fit_knn, validation)

 rmse_acc <- rmse(validation$rating,edx_hat_knn)

 })

max(accurac)
```

ks[is.max(accurac)]

Finally, running a knn with k = 3 we obtain the following results

```
fit_knn <- knn3(rating ~ userId+movieId+timestamp+genresId, data = edx_prc, k = 3)

 edx_hat_knn <- predict(fit_knn, validation)

 rmse_acc <- rmse(validation$rating,edx_hat_knn)
```

```
##############################################################################
```

###Third method: classification tree

The third method presented is a classification tree. This method was able to provide a better RMSE.

```
fit_rt <- rpart(rating ~ . , data = edx)

edx_hat_rt <- predict(fit_rt, validation)

rmse_acc <- rmse(validation$rating,edx_hat_rt)
```

Appendix 1:

Code used in several techniques in data wrangling.

Title extraction

```
edx <- edx %>% extract(title, c("title_noY", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F)
```

Timestamp into date format (via lubridate function)

```
edx$timestamp <- as_datetime(edx$timestamp)

edx$year <- as.numeric(edx$year)

edx <- edx %>% mutate(year_rat = year(timestamp))
```

Genres wrangling

```
gnr <- edx[,c(2,8)]

gnr <- unique(gnr)

gnr <- gnr %>% separate_rows(genres, sep = "\\|")

gnr$genres <- as.character(gnr$genres)

binary_gnr <- dummyVars(~ genres, data = gnr)

gnr2 <- predict(binary_gnr, gnr)

head(gnr2)

names(gnr)

head(binary_gnr$facVars)

gnr2 <- data.frame(gnr2)

gnr2 <- gnr2 %>% mutate(movieId = gnr$movieId)

gnr2 <- gnr2 %>% group_by(movieId) %>% summarise(

  genres.no.genres.listed = sum(genres.no.genres.listed.),

  genresAction = sum(genresAction),

  genresAdventure = sum(genresAdventure),

  genresAnimation = sum(genresAnimation),

  genresChildren = sum(genresChildren),
```

```
  genresComedy = sum(genresComedy),
  genresCrime = sum(genresCrime),
  genresDocumentary = sum(genresDocumentary),
  genresDrama = sum(genresDrama),
  genresFantasy = sum(genresFantasy),
  genresFilm.Noir = sum(genresFilm.Noir),
  genresHorror = sum(genresHorror),
  genresIMAX = sum(genresIMAX),
  genresMusical = sum(genresMusical),
  genresMystery = sum(genresMystery),
  genresRomance = sum(genresRomance),
  genresSci.Fi = sum(genresSci.Fi),
  genresThriller = sum(genresThriller),
  genresWar = sum(genresWar),
  genresWestern = sum(genresWestern),
)
remove(binary_gnr, gnr)
edx <- edx %>% left_join(gnr2, by = "movieId")
edx <- edx %>% left_join(gnr2, by = 'movieId')
```

Note: the same transformation is applied to validation dataset