Movielens Capstone Project

Aurelius Ferdinand Almeida

1/9/2020

Introduction

This analysis is done for a capstone project submission as part of the Harvard Data science Professional Certification. The dataset used is the Movielens dataset created by Grouplens. This is the 10 Million record version of the dataset and contains millions of ratings provided by users for a large set of movies. A standard code has been provided that procures the data from the grouplens website, formats and partitions it into a training set named 'edx' with 9,000,055 records and a test set named 'validation' containing 999,999 records. The code can be found in Methods and Analysis below.

-

  Objectives

  The key objective of this project is to successfully train a machine learning model on the edx dataset and then predict ratings on the validation dataset with an **RMSE <= 0.8649**.

-

  Summary

  The project has provided multiple learning avenues and also presented several challenges to be overcome. The methods and analysis section elaborates these points further. Models that were applied but were not successful in meeting the required RMSE threshold have been discussed in brief. The final model chosen for this purpose was built using the recosystem package. It achieved a **RSME of 0.78267** and is detailed in the results section. The report concludes with notes on the project, its limitations and possible future work.

Methods and Analysis

-

  Getting Started

  Any analysis requires the right tools to be made available. In R these are the additional libraries and packages, the code chunk below will install if needed and load the required libraries

```r
#Installing if necessary and loading the required packages
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")
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
if(!require(foreach)) install.packages("foreach", repos = "http://cran.us.r-project.org")
if(!require(MASS)) install.packages("MASS", repos = "http://cran.us.r-project.org")
if(!require(gam)) install.packages("gam", repos = "http://cran.us.r-project.org")
if(!require(pls)) install.packages("pls", repos = "http://cran.us.r-project.org")
```

The standard code provided to generate the edx and validation datasets is below.

```r
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

# 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")
```

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

```r
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

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

- **Overview of the dataset**

The edx(train) and validation(test) datasets have a similar structure and format with a differing number of observations.The characteristics of the dataset can be viewed by using the code below

```
# View structure of the dataset
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 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
```

The summary function will provide additional useful information regarding the dataset. The first 5 rows of the dataset can be viewed using the below code

```
#View summary of the dataset
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# View first 5 rows of the dataset

head(edx,5)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                         genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

The dataset has 6 variables of which 'rating' is the variable to be predicted and 'userId' and 'movieId' are the primary predictors. From the remaining variables genres and timestamp are also possible predictors to be used. An immediate observation from this dataset is the number of records, at just over 9 million it is anticipated that there will be a substantial computational burden with model application using a standard laptop.Delving further into the dataset the summarise function helps to identify the number of unique users and movies

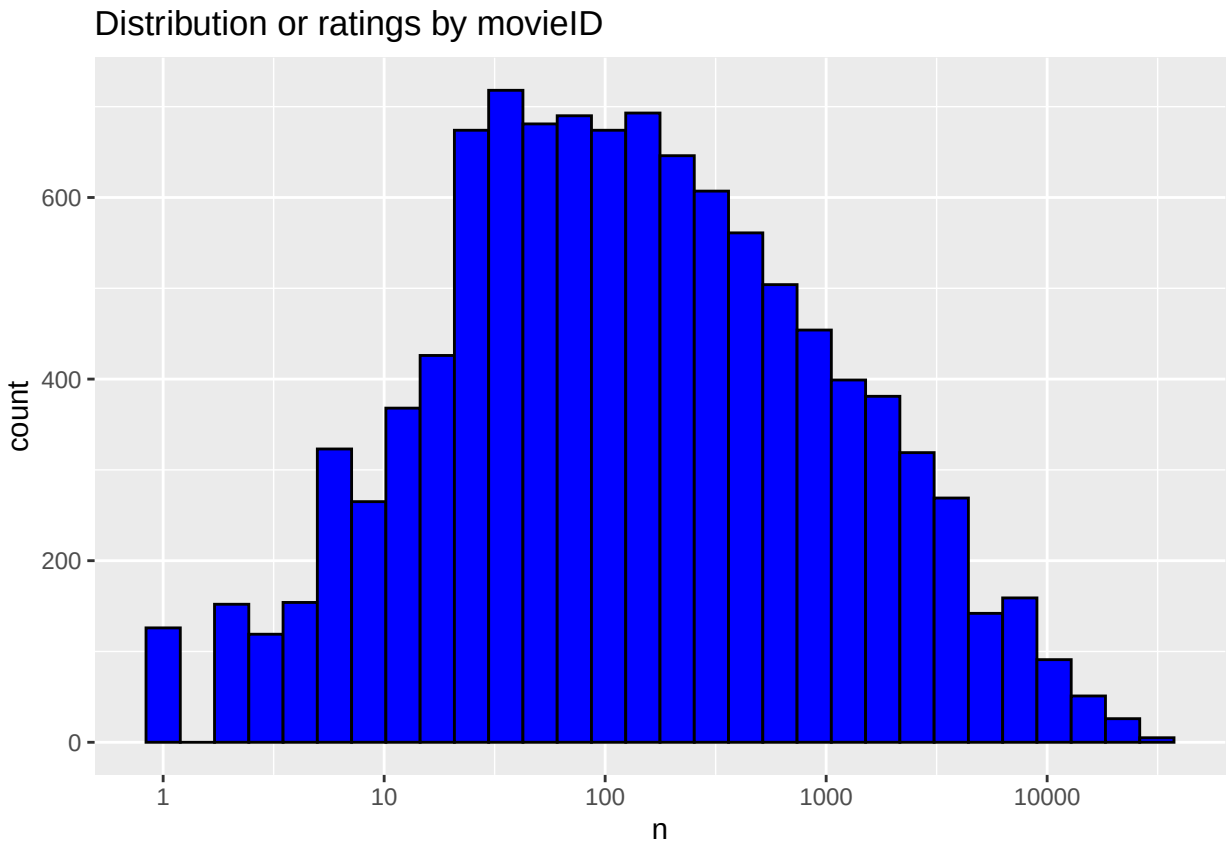
```
# identify unique users and movies
edx %>%
  summarize(users = n_distinct(userId),
            movies = n_distinct(movieId))
```

```
##   users movies
## 1 69878  10677
```

When multiplied 10677 unique movies and 69878 unique users the number of expected ratings would be several order of magnitudes larger that the current number of records. This indicates that not all movies have been rated and not all users may have provided ratings. A visualisation of these distributions can shed more light on this point

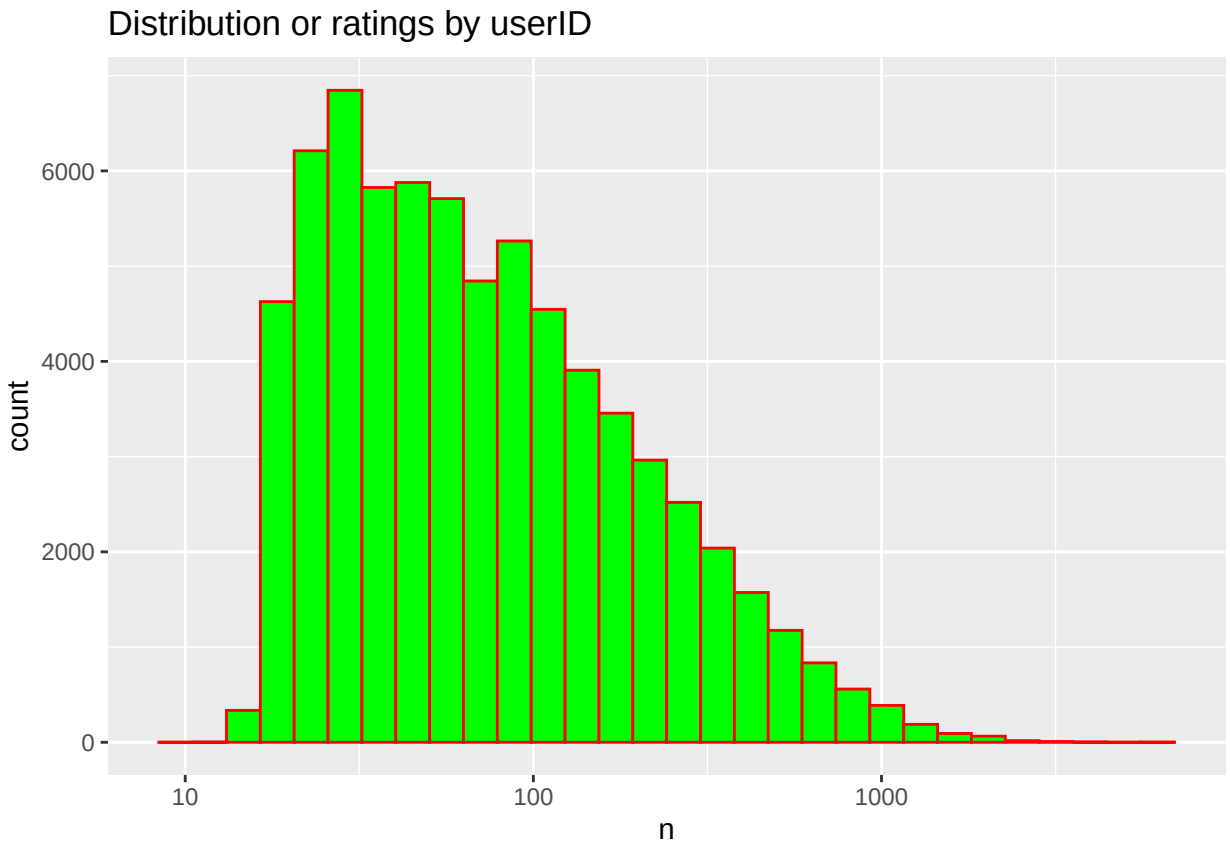
```
# plot distribution of ratings by movieId

edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, fill = "blue", color = "black") +
  scale_x_log10()+
  ggtitle("Distribution or ratings by movieID")
```



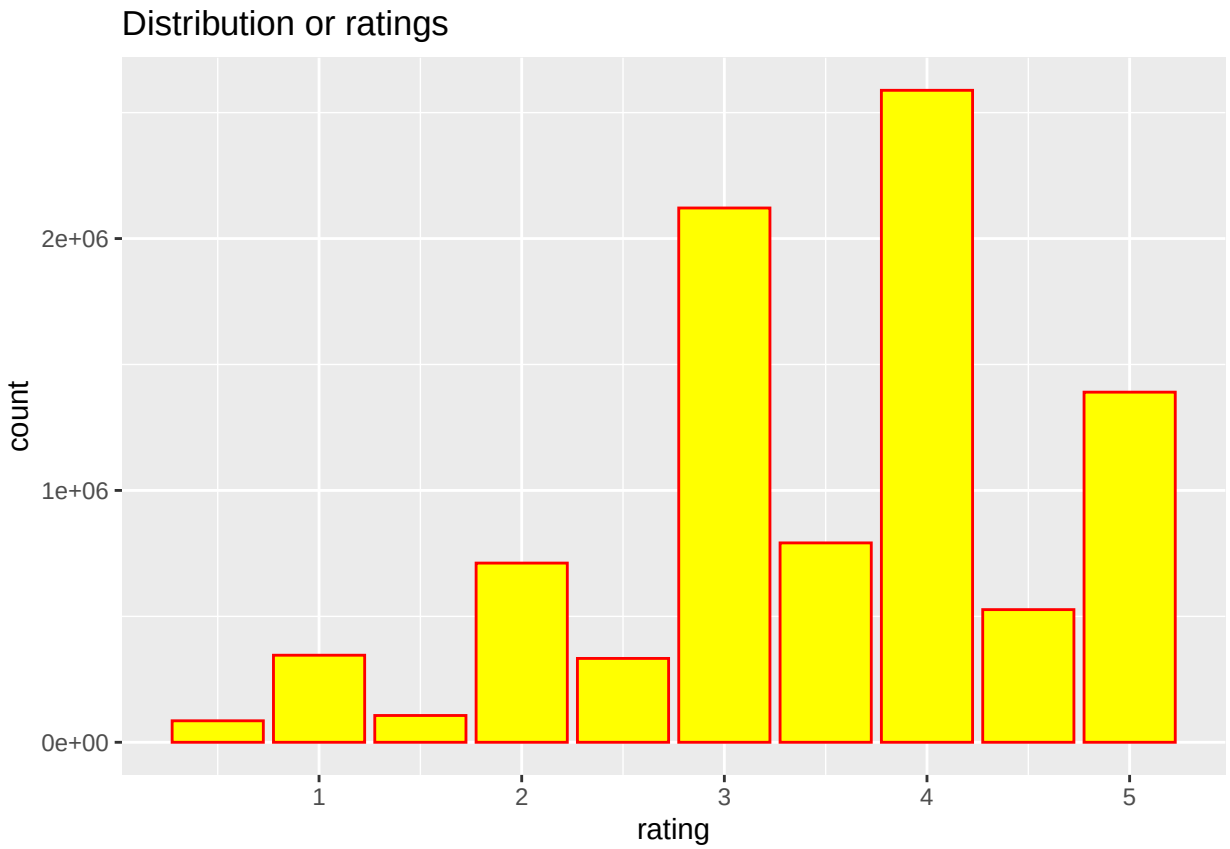

```
# plot distribution of ratings by userId

edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, fill = "green", color = "red") +
  scale_x_log10()+
  ggtitle("Distribution or ratings by userID")
```



Analysis of these distributions indicate that ratings distribution among movies is unequal with some movies getting rated far more than others. A similar case is also seen in the distribution of ratings by userId, some users have provided a large number of ratings while there are others with almost none.The rating distribution in itself can be visualised using a simple bar plot as below.

```
# Using a bar plot to visualise rating distribution
edx %>% group_by(rating) %>%
  ggplot(aes(rating)) +
  geom_bar(fill = "yellow", color = "red") +
    ggtitle("Distribution or ratings")
```



The plot shows that are there are 10 ratings values from 0.5 to 5 which correspond with the system of 5 star ratings used to grade movies by users. In general there are more average to good ratings than there are poor, also full star ratings are more likely than half star ratings.

- **Creating the function to test RMSE**

  A standardised function will be used to evaluate the root mean squared error of the model and will be created using the below provided code

```
# Create funtion that will calculate RMSE
mvl_RMSE <- function(true_ratings,predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2))
}
```

- **Application of Machine learning models**

In chapter 33.7 of the book "Introduction to Data science", Prof Rafael Irizarry has explained recommendation systems with an example of the Netflix challenge and a subset of the movielens dataset. Estimates were generated using summary calculations that indicated using user effects and movie effects in the model can achieve the desired RMSE. The analysis will consider these assumptions and proceed. The validation

dataset will be reserved only for the final test of RMSE, the edx dataset will be used for all initial training and testing purposes. Given the number of observations in the edx dataset, it is advisable to first test models on a smaller subset of the dataset before applying it to the larger context.

At first a subset of the edx dataset is created, then using a function from the caret package it will be partitioned into an initial train and test set as below.

```
# Set seed and Create a subset
set.seed(10, sample.kind = "Rounding")
```

```
## Warning in set.seed(10, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
edxsub <- edx %>% sample_n(50000)

# Create data partition
index <- createDataPartition(edxsub$rating,times = 1, p = 0.8, list = FALSE)

# Use index to create train and test objects
mvltrain <- edxsub[index,]
mvltest <- edxsub[-index,]
mvltest <- mvltest %>%
  semi_join(mvltrain, by = "movieId") %>%
  semi_join(mvltrain, by = "userId")
```

Once the subsets are created the next step is to choose the machine learning model to be applied. The first model applied will be 'Partial least squares' from the pls package. The model will be called from within the train function of caret which support 238 models at this time. More information is available in this link. Prior to training the model the control and tuning parameters will be defined

```
# Create the control and grid that the training model can use as inputs

control <- trainControl(method = "cv",number = 10, p=0.1)
grid <- data.frame(ncomp = c(1,2))
```

Once this is done the machine learning model will be trained on the mvltrain subset of edx dataset that was created in the previous step.

```
# Train the machine learning model on the training subset of the edx dataset
plst <- train(rating~movieId+userId, data = mvltrain,
              method  = "pls",
              trControl = control,
              tuneGrid = grid,
              preProcess = "scale",
              allowParallel = TRUE)
```

The trained algorithm will then be applied on the test subset of the edx dataset for predictions.

```
# Apply the trained model on the test subset of the edx dataset
yhatplst <- predict(plst,mvltest)
```

Once the predictions are available they can be called into the function provided to calculate RMSE

```r
# Calculate RMSE using the function provided
mvl_RMSE(mvltest$rating,yhatplst)
```

```
## [1] 1.056402
```

The RMSE calculation achieved was much higher than the threshold set for the project. Changing tuning and control parameters were unsuccessful at significantly reducing the calculated RMSE. Additional models were applied using a similar methodology. These are 'Principal component analysis' from pls package, 'Robust Linear Model' from the MASS package and 'Generalized Additive Model using LOESS' from the gam package. All models were trained using the train function from caret. The next step taken was to increase the sample size progressively to determine if this change would lead to a significant reduction in RMSE value. However a significant challenge experienced here was the limitation of RAM memory. Using over half of the edx dataset observations in these very quickly started to exhaust system memory and led to system hang ups and crashes. A memory upgrade from 8 to 16 gb RAM solved the problem and allowed use of larger subset of observations however no decrease in RMSE value was noted in any of these runs.

Early experimentation with k nearest neighbor and random forest models were abandoned as even on relatively small sample sizes the processing time needed and system memory needed were very large. It was deemed highly improbable that such models could be applied successfully on the larger observation set given the computational limitations of a standard laptop. It was clear a different method was necessary to achieve the target RMSE reduction. Matrix factorisation was then explored as a possible solution. Research into this area led to the discovery of the recosystem package. A review of the package documentation suggested that this could indeed solve both the problems of RAM memory usage and the required RMSE reduction.This package uses matrix factorisation as a model and stores objects on disk instead of keeping them in memory. This prevents available RAM memory from being exhausted even when working on very large observations. It also takes advantage of multicore processing and can significantly shorten the time required for model training. The successful implementation of this model on the complete dataset is discussed in detail in the results section.

Results

The application of the recosystem package allowed the analysis to meet the objective of reducing RMSE. The calculated RMSE obtained was 0.78267 and the steps followed are described here. The first step needed is to only keep the columns required for the analysis in both datasets, rename the columns and convert these datasets to a matrix.

```r
# The recosystem package applying matrix factorisation
# Step 1; SUbset edx and validation retain only the required 3 fields, rename and create matrix

edx1 <-  edx %>% dplyr::select("userId","movieId","rating")
valid1 <- validation %>% dplyr::select("userId","movieId","rating")

# Rename columns in the newly subsetted files
names(edx1) <- c("user", "item", "rating")
names(valid1) <- c("user", "item", "rating")

# Change the files to a matrix
edx1 <- as.matrix(edx1)
valid1 <- as.matrix(valid1)
```

The files are then written in tables onto the hard disk

```
# Step 2; Write the matrix files to tables on disk
write.table(edx1, file = "train.txt", sep = " ", row.names = FALSE, col.names = FALSE)
write.table(valid1, file = "test.txt", sep = " ", row.names = FALSE, col.names = FALSE)
```

Post which data source objects are created that are linked to files written on the disk

```
# Step 3; Use data_file to create a data source object that links the written file

dir <- getwd()
train <- file.path(dir,"train.txt")
test <- file.path(dir,"test.txt")

# Use set seed before running data_file
set.seed(10, sample.kind = "Rounding")
```

```
## Warning in set.seed(10, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
train_1 <- data_file(train)
test_1 <- data_file(test)
```

Once the data source objects are available proceed with creating a reco recommender object and apply tuning parameters. It is recommended to review this link for parameters needed. The tuning function takes the most amount of time needed however compared to other models previously gaged it runs on the complete training observations in under 15 minutes. Once run it also creates a horizontal completion bar in the console window that starts to fill in as the calculations proceed. It was found to be a very useful feature of this package.

```
# Step 4; Create the reco recommender object and applying tuning parameters to the model

r <- Reco()

# Apply tuning parameters

tparam <- r$tune(train_1, opts = list(dim = c(10, 20, 30),
                                      lrate = c(0.1, 0.2),
                                      costp_l1 = 0,
                                      costq_l1 = 0,
                                      nthread = 3,
                                      niter = 10))
tparam
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
```

```
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.7973338
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8257659
## 2   20        0     0.01        0     0.01   0.1 0.8082276
## 3   30        0     0.01        0     0.01   0.1 0.8148946
## 4   10        0     0.10        0     0.01   0.1 0.8278937
## 5   20        0     0.10        0     0.01   0.1 0.8022497
## 6   30        0     0.10        0     0.01   0.1 0.8007592
## 7   10        0     0.01        0     0.10   0.1 0.8299321
## 8   20        0     0.01        0     0.10   0.1 0.8031223
## 9   30        0     0.01        0     0.10   0.1 0.7973338
## 10  10        0     0.10        0     0.10   0.1 0.8383637
## 11  20        0     0.10        0     0.10   0.1 0.8296686
## 12  30        0     0.10        0     0.10   0.1 0.8285392
## 13  10        0     0.01        0     0.01   0.2 0.8157163
## 14  20        0     0.01        0     0.01   0.2 0.8212485
## 15  30        0     0.01        0     0.01   0.2 0.9246798
## 16  10        0     0.10        0     0.01   0.2 0.8641837
## 17  20        0     0.10        0     0.01   0.2 0.9107646
## 18  30        0     0.10        0     0.01   0.2 0.9604459
## 19  10        0     0.01        0     0.10   0.2 0.8206653
## 20  20        0     0.01        0     0.10   0.2 0.8008582
## 21  30        0     0.01        0     0.10   0.2 0.7996557
## 22  10        0     0.10        0     0.10   0.2 0.8397854
## 23  20        0     0.10        0     0.10   0.2 0.8253883
## 24  30        0     0.10        0     0.10   0.2 0.8241378
```

After the parameter tuning is completed the model can be trained and predictions made on the test file.

```
# Step 5; Train the recommender model on the train file and make predictions using test

r$train(train_1, opts = c(tparam$min, nthread = 5, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9732  1.2042e+007
##    1       0.8723  9.8828e+006
##    2       0.8390  9.1731e+006
##    3       0.8174  8.7559e+006
##    4       0.8020  8.4828e+006
##    5       0.7903  8.2816e+006
```

```
##    6       0.7807  8.1315e+006
##    7       0.7725  8.0105e+006
##    8       0.7656  7.9162e+006
##    9       0.7595  7.8321e+006
##   10       0.7542  7.7620e+006
##   11       0.7495  7.7044e+006
##   12       0.7452  7.6539e+006
##   13       0.7414  7.6074e+006
##   14       0.7379  7.5703e+006
##   15       0.7346  7.5354e+006
##   16       0.7317  7.5024e+006
##   17       0.7290  7.4755e+006
##   18       0.7265  7.4484e+006
##   19       0.7242  7.4239e+006
```

```
preds = tempfile()

r$predict(test_1, out_file(preds))
```

```
## prediction output generated at C:\Users\aurel\AppData\Local\Temp\RtmpC2eXHT\file4a841f1274c
```

The last step in the process requires objects to be created to hold both the actual results and the predicted results. Once this is done the provided function can then be used to calculate the RMSE of the model.

```
# Step 6; Assign the actual and predicted ratings to objects and calculate RMSE

actualrat <- read.table("test.txt", header = FALSE, sep = " ")$V3
predrat <- scan(preds)

reco_RMSE <- mvl_RMSE(actualrat, predrat)
reco_RMSE
```

```
## [1] 0.7825169
```

Conclusions

Application of the recosystem package that is primarily designed as a recommendation system allows the analysis to meet the objective of achieving a root mean square error RMSE of < 0.8649. While this objective is met the model does have limitations as it does not take into account the genre effect. While much lower there is also a effect of ratings over time that this model cannot account for. Future work would require both these predictors to be included. While not present in this dataset there could also be a potential effect of cast on rating predictions and would be worth exploring.

This project analysis has provided several avenues to apply data science skills and techniques while presenting challenges to be overcome. It has been an invaluable learning experience.

This report was created in R using the R Studio IDE and the R Markdown package