**Project Report**

on

**Biblio Legio**

**Book Recommendation System**

**Submitted**

by

**Aditi Namboodiripad**

**Dalia Joseph**

**I P Raghesh Kumar**

as part fulfillment of the

**IBM ICECSP Global Certification Programme**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sl No | Particulars | Page |
| 1 | Executive Summary | 3 |
| 2 | Introduction | 4 |
| 3 | System, Implementation & Testing | 6 |
| 4 | Conclusion | 23 |
| 5 | Further research | 24 |
| 6 | References | 25 |
| 7 | Appendix | 26 |

**Executive Summary**

Book recommendation systems are widely used to recommend the best reads that are most appropriate to book lovers. They provide benefits to both the reader and the seller, by suggesting books to the readers, which motivates and increases the demand for the recommended items.The recommender system consists of two entities, the readers and the books. The input to this book recommendation algorithm is a database of readers and books and output is the book recommendations.

This book recommendation which we have termed ‘Biblio Legio’ is prepared using R Programming and helps to buy the books that are of the buyer’s interest.This system uses features of collaborative filtering to produce efficient and effective recommendations through R’s Recommender lab. It aggregates the ratings of books, recognises commonalities between readers based on their ratings and generates new recommendations. Collaborative recommendation is probably the most familiar, most widely implemented and most mature of the technologies. The dataset for the book recommendation was taken from Goodreads. Further R’s Shiny software forms the GUI, presenting the recommendations beautifully.

**Introduction**

**1. Background**

Recommender systems can be implemented in any domain from E-commerce to network security in the form of personalized services. They provide benefits to both the consumer and the seller, by suggesting items to consumers, which motivates and increases the demand for the recommended items.The book recommendation will help the reader to buy the books that they would love reading as the algorithm will present to them books that are of their interest.

**1.1 Aim of the Project**

The project aims to provide a Book Recommendation System based on Collaborative filtering which will

* Save the precious time of the customer and is very efficient to use.
* Provides a large number of choices for books & also recommends books.
* Users can buy books easily by making online payments.
* Usage of ML ensures reduction of human bias in selection, classification and recommendation
* The system recommending algorithm scales well with co-rated items.

**1.2 Technologies Used:**

The project was done with R Studio’s Recommender lab which uses the popular Collaborative filtering (CF) algorithm used for making recommendations. For classification we have used the KNN algorithm, one of the most popular Machine learning algorithms. For creating the GUI we have used R Shiny and also incorporated Java Script and CSS.

**1.2.1 Literature Review**

*Recommendation System:* A recommendation system is an information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.**[1]** Recommender systems are utilized in a variety of areas and are most commonly recognized as playlist generators for video and music services, product recommenders for online stores, or content recommenders for social media platforms.**[2]** These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries.**[3]** There are also popular recommender systems for specific topics like restaurants and online dating. Recommender systems have also been developed to explore research articles and experts,**[4]** collaborators,**[5]** and financial services.

*Collaborative filtering (CF):* It is a technique used by recommender systems**[1]**. Collaborative filtering has two senses, a narrow one and a more general one.**[6]**

In the narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person**[7]**

In the more general sense, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.**[6]** Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc.

*Machine learning (ML):* It is the study of computer algorithms that improve automatically through experience.**[8]** It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, in order to make predictions or decisions without being explicitly programmed to do so.**[9**[**]**](https://en.wikipedia.org/wiki/Machine_learning#cite_note-2) Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks. As it is applied across business problems, machine learning is also referred to as predictive analytics.

*k-nearest neighbors algorithm (k-NN):* is a non-parametric method used for classification and regression.**[10]** In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive [i](https://en.wikipedia.org/wiki/Integer)nteger, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

**System, Implementation & Testing**

**2.1 Requirements**

The following are the different requirements for the recommender to be successful. It includes hardware, software, functional and user requirements.

**2.1.1 Hardware Requirements**

* i3 Processor Based Computer or higher
* Memory: 1 GB RAM
* Hard Drive: 50 GB
* Monitor
* Internet Connection
* Mouse

**2.1.2 Software Requirements**

* Windows 7 or higher
* R Studio
* R Tools
* R Version 4.0.3
* Google Chrome

**2.1.1 Functional requirements**

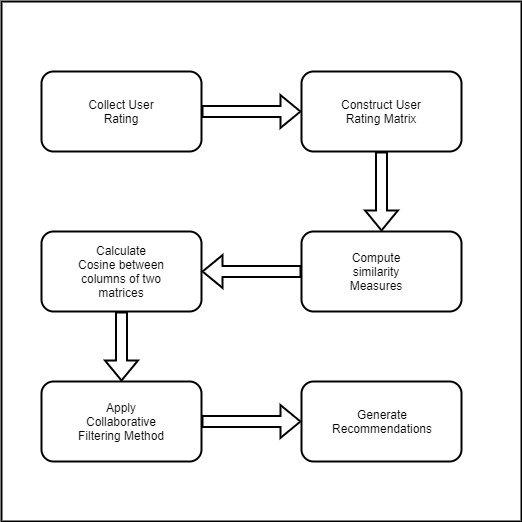
The recommender should be able to give correct recommendations based on the ratings given by other similar users. A large choice of books is a must have requirement. The recommender should be easy to use. It should also provide the user with an option to buy the books so recommended.

**2.1.2 User Requirements**

Users should have a compatible computer/laptop/tablet or mobile and internet connection. The user has to at least rate one book to get proper recommendations.

**2.2 Design and Architecture**

**Figure 1**



The process flow of the project is as per Figure 1 given above

**2.3 Implementation**

**#Implementation of collaborative filtering.**

library(Matrix)

library(recommenderlab)

library(slam)

library(data.table)

#First we calculate rating predictions

calculate\_predictions <- function(ratings\_matrix, similarity\_matrix)

# We use this function to determine similarities with user rating, using knn approach

find\_similarities <- function(matrix, columns\_to\_consider, similarity\_metric, make\_positive\_similarities, k){

selected\_columns <- matrix[, columns\_to\_consider, drop=FALSE]

#similarities should be dgCMatrix with explicit zeros in places where similarity is zero.

similarities <- similarity\_metric(matrix, selected\_columns)

#In order to keep explicit zeros, change them to some value close to zero.

#Then we will set similarities of users/item to themselves to zero and drop those values.

similarities@x [similarities@x == 0] <- 0.000001

ind <- cbind(columns\_to\_consider, 1:length(columns\_to\_consider))

similarities[ind] <- 0

similarities <- drop0(similarities)

#Make all similarities positive,

if(make\_positive\_similarities) {

if(min(similarities@x) < 0) similarities@x <- similarities@x + abs(min(similarities@x))

}

if(!is.null(k) && k < nrow(similarities) - 1) {

#if ALL - 1 that means we need all neigbours except that user/item

#find k nearest neighbours by using slam::simple\_triplet\_matrix and data.table.

# Save dim and dimnames, in order to later reconstruct from simple\_triplet\_matrix.

dims\_old <- similarities@Dim

dimnames\_old <- similarities@Dimnames

similarities <- as.simple\_triplet\_matrix(similarities)

datatable <- data.table(similarities$i, similarities$j, similarities$v)

names(datatable) <- c("row", "column", "rating")

#Function that finds k-th largest value in a vector.

kthMax <- function(vector, k){

if(length(vector) <= k) min(vector)

else{

sort(vector, partial = length(vector) - (k-1))[length(vector) - (k-1)]

}

}

kthMaxes <- datatable[, kthMax(rating, k), by = column]

names(kthMaxes) <- c("column", "kthMax")

datatable <- merge(datatable, kthMaxes, by="column")

datatable <- datatable[datatable$rating >= datatable$kthMax, ]

similarities <- as(sparseMatrix(i = datatable$row, j = datatable$column, x = datatable$rating, dims = dims\_old, dimnames = dimnames\_old), "dgCMatrix")

}

similarities

}

# we use this function to add the determined predictions to prediction matrix

add\_predictions\_to\_prediction\_matrix <- function(predictions\_matrix, part\_predictions, predictions\_matrix\_indices){

row\_names <- as.integer(unlist(part\_predictions@Dimnames[1])) # Real row indices from predictions matrix.

columns\_names <- as.integer(unlist(part\_predictions@Dimnames[2]))

row\_info <- cbind(row\_name = row\_names, row\_index = 1:length(row\_names)) # row\_index = row indices from part\_predictions.

column\_info <- cbind(column\_name = columns\_names, column\_index = 1:length(columns\_names))

all\_indices <- predictions\_matrix\_indices

colnames(all\_indices) <- c("row\_name", "column\_name")

all\_indices <- merge(all\_indices, row\_info)

all\_indices <- merge(all\_indices, column\_info)

predictions\_matrix\_indices <- all\_indices[, c("row\_name", "column\_name")]

part\_matrix\_indices <- all\_indices[, c("row\_index", "column\_index")]

if(nrow(predictions\_matrix\_indices) > 0){

predictions\_matrix[as.matrix(predictions\_matrix\_indices)] <- part\_predictions[as.matrix(part\_matrix\_indices)]

}

predictions\_matrix

}

#Predition matrix is obtained

#This function implements memory-based collaborative filtering and calculates rating predictions.

#It divides matrix into parts and calcualtes predictions for each part iteratively.

#This can be useful in case matrices are large and can not fit into memory.

predict\_cf <- function(ratings\_matrix, predictions\_indices, alg\_method, normalization, similarity\_metric, k, make\_positive\_similarities, rowchunk\_size, columnchunk\_size){

if(normalization){

# Currently, we always use center normalization and apply it per users (subtracting user averages).

if(alg\_method == "ubcf") ratings\_matrix <- normalize(as(ratings\_matrix, "realRatingMatrix"), method = "center", row = FALSE)

if(alg\_method == "ibcf") ratings\_matrix <- normalize(as(ratings\_matrix, "realRatingMatrix"), method = "center", row = TRUE)

ratings\_matrix@data@x[ratings\_matrix@data@x == 0] <- 0.000001 # Prevent droping zeros obtained after applying normalization.

normalization\_info <- ratings\_matrix@normalize

ratings\_matrix <- as(ratings\_matrix, "dgCMatrix")

}

# Create initial empty predictions matrix.

predictions\_matrix <- as(sparseMatrix(i = c(), j = c(), dims = ratings\_matrix@Dim, dimnames = ratings\_matrix@Dimnames), "dgCMatrix")

# Number of splits per rows and columns.

num\_row\_splits <- ceiling(nrow(ratings\_matrix)/rowchunk\_size)

num\_column\_splits <- ceiling(ncol(ratings\_matrix)/columnchunk\_size)

# Iterate over columns first, so that each chunk of similarities is calcualated only once.

for(i in 1:num\_column\_splits){

start\_column <- columnchunk\_size \* (i-1) + 1 # Start column for the current chunk.

end\_column <- columnchunk\_size \* i # End column for the current chunk.

if(ncol(ratings\_matrix) < end\_column){

end\_column <- ncol(ratings\_matrix)

}

columns\_to\_consider <- intersect(start\_column:end\_column, predictions\_indices[, 2])

if(length(columns\_to\_consider) == 0) next

# Set names of rows and columns to be numbers (indices).

# This way similarities and part\_predictions, calculated in next steps, will use these names.

ratings\_matrix@Dimnames[[1]] <- as.character(1:nrow(ratings\_matrix))

ratings\_matrix@Dimnames[[2]] <- as.character(1:ncol(ratings\_matrix))

similarities <- find\_similarities(ratings\_matrix, columns\_to\_consider, similarity\_metric, make\_positive\_similarities, k)

for(j in 1:num\_row\_splits){

start\_row <- rowchunk\_size \* (j-1) + 1 # Start row for the current chunk.

end\_row <- rowchunk\_size \* j # End row for the current chunk.

if(nrow(ratings\_matrix) < end\_row){

end\_row <- nrow(ratings\_matrix)

}

rows\_to\_consider <- intersect(start\_row:end\_row, predictions\_indices[, 1])

if(length(rows\_to\_consider) == 0) next

# print(paste("Current chunk: ", start\_row, end\_row, start\_column, end\_column, sep = ","))

part\_predictions <- calculate\_predictions(ratings\_matrix[rows\_to\_consider, , drop = FALSE], similarities) # drop = FALSE because of the case when we have only one row, make it dgCMatrix.

# Fill predictions matrix with predictions calculated in this iteration.

predictions\_indices\_to\_consider <- subset(predictions\_indices, predictions\_indices[, 1] %in% rows\_to\_consider & predictions\_indices[, 2] %in% columns\_to\_consider)

predictions\_matrix <- add\_predictions\_to\_prediction\_matrix(predictions\_matrix, part\_predictions, predictions\_indices\_to\_consider)

}

}

if(normalization){

temp <- as(predictions\_matrix, "realRatingMatrix")

temp@normalize <- normalization\_info

predictions\_matrix <- denormalize(temp)

predictions\_matrix <- as(predictions\_matrix, "dgCMatrix")

}

predictions\_matrix

}

predict\_cf <- function(ratings\_matrix, predictions\_indices, alg\_method, normalization, similarity\_metric, k, make\_positive\_similarities, rowchunk\_size, columnchunk\_size){

if(normalization){

if(alg\_method == "ubcf") ratings\_matrix <- normalize(as(ratings\_matrix, "realRatingMatrix"), method = "center", row = FALSE)

if(alg\_method == "ibcf") ratings\_matrix <- normalize(as(ratings\_matrix, "realRatingMatrix"), method = "center", row = TRUE)

ratings\_matrix@data@x[ratings\_matrix@data@x == 0] <- 0.000001 # Prevent droping zeros obtained after applying normalization.

normalization\_info <- ratings\_matrix@normalize

ratings\_matrix <- as(ratings\_matrix, "dgCMatrix")

}

# empty predictions matrix.

predictions\_matrix <- as(sparseMatrix(i = c(), j = c(), dims = ratings\_matrix@Dim, dimnames = ratings\_matrix@Dimnames), "dgCMatrix")

# Number of splits per rows and columns.

num\_row\_splits <- ceiling(nrow(ratings\_matrix)/rowchunk\_size)

num\_column\_splits <- ceiling(ncol(ratings\_matrix)/columnchunk\_size)

# We have iterated over columns first, so that each chunk of similarities is calcualated only once.

for(i in 1:num\_column\_splits){

start\_column <- columnchunk\_size \* (i-1) + 1

end\_column <- columnchunk\_size \* i

if(ncol(ratings\_matrix) < end\_column){

end\_column <- ncol(ratings\_matrix)

}

columns\_to\_consider <- intersect(start\_column:end\_column, predictions\_indices[, 2])

if(length(columns\_to\_consider) == 0) next

# We have set names of rows and columns to be numbers

ratings\_matrix@Dimnames[[1]] <- as.character(1:nrow(ratings\_matrix))

ratings\_matrix@Dimnames[[2]] <- as.character(1:ncol(ratings\_matrix))

similarities <- find\_similarities(ratings\_matrix, columns\_to\_consider, similarity\_metric, make\_positive\_similarities, k)

for(j in 1:num\_row\_splits){

start\_row <- rowchunk\_size \* (j-1) + 1

end\_row <- rowchunk\_size \* j

if(nrow(ratings\_matrix) < end\_row){

end\_row <- nrow(ratings\_matrix)

}

rows\_to\_consider <- intersect(start\_row:end\_row, predictions\_indices[, 1])

if(length(rows\_to\_consider) == 0) next

part\_predictions <- calculate\_predictions(ratings\_matrix[rows\_to\_consider, , drop = FALSE], similarities) # drop = FALSE because of the case when we have only one row, make it dgCMatrix.

# Fill predictions matrix with predictions calculated in this iteration.

predictions\_indices\_to\_consider <- subset(predictions\_indices, predictions\_indices[, 1] %in% rows\_to\_consider & predictions\_indices[, 2] %in% columns\_to\_consider)

predictions\_matrix <- add\_predictions\_to\_prediction\_matrix(predictions\_matrix, part\_predictions, predictions\_indices\_to\_consider)

}

}

if(normalization){

temp <- as(predictions\_matrix, "realRatingMatrix")

temp@normalize <- normalization\_info

predictions\_matrix <- denormalize(temp)

predictions\_matrix <- as(predictions\_matrix, "dgCMatrix")

}

predictions\_matrix

}

**# Similarity measures for sparse matrices.**

#Calculates the correlations between columns of two sparse matrices.

cal\_cor <- function(X, Y){

availX <- X!=0

availY <- Y!=0

# normalization

X<- as(normalize(as(X, "realRatingMatrix"), method = "center", row = FALSE), "dgCMatrix")

Y<- as(normalize(as(Y, "realRatingMatrix"), method = "center", row = FALSE), "dgCMatrix")

R <- crossprod(X,Y)

N <- crossprod(X^2, availY)

M <- crossprod(availX, Y^2)

cor <- R

cor@x <- cor@x/((N@x^0.5) \* (M@x^0.5))

cor #correlation matrix is returned

}

# Calculates cosine between columns of two sparse matrices.

cal\_cos <- function(X, Y){

ones <- rep(1,nrow(X))

means <- drop(crossprod(X^2, ones)) ^ 0.5

diagonal <- Diagonal( x = means^-1 )

X <- X %\*% diagonal

ones <- rep(1,nrow(Y))

means <- drop(crossprod(Y^2, ones)) ^ 0.5

diagonal <- Diagonal( x = means^-1 )

Y <- Y %\*% diagonal

crossprod(X, Y) #Returns the matrix of cosine values

}

**2.4 Testing**

Proper testing is very important when coding for any project. Below are the details of the testing done:

**2.4.1 Test Plan Objectives**

To ensure that the Book Recommender is error free we have to execute the program with the aim of finding errors.

1. The recommender should meet the reader’s requirements
2. Ideally the testing should be done by a independent assessor

**2.4.2 Data Entry**

We have taken the data of Goodreads from Kaggle and done data cleaning. The cleaned data set has been used for the project.

**2.4.3 Test Strategy**

We believe that the test strategy should be as follows:

1. Optimal instead of exhaustive testing is what is suggested.
2. Usually errors follow the Pareto (80-20) rule of 80% error coming from 20% of programme components
3. It is imperative that we cover the basic tests before going to complex ones.

**2.4.4 System Test**

It works on Windows 7 as well as OS X 10.11 El Capitan and their respective higher versions. Further tests on Linux operating systems need to be done.

**2.4.5 Performance Test**

We have used the R Programmes profiler to run the performance test. The following are the results for the different code components:

* Server: 20740 ms
* UI: 15600 ms
* CF Algorithm: 9740 ms
* Similarity Measures: 10500 ms
* Helpers: 59220 ms

R Programming has a drawback. The CPU cycles increases as the time R Studio is kept open increases.

**2.4.6 Security Test**

Security tests are yet to be conducted.

**2.4.7 Basic Test**

We have divided the code into five distinct components

1. Server
2. UI
3. CF algorithm
4. Similarity measures
5. Helpers (Try catch statement, Java Script for buttons etc.)

By using a sample input we have checked and verified that

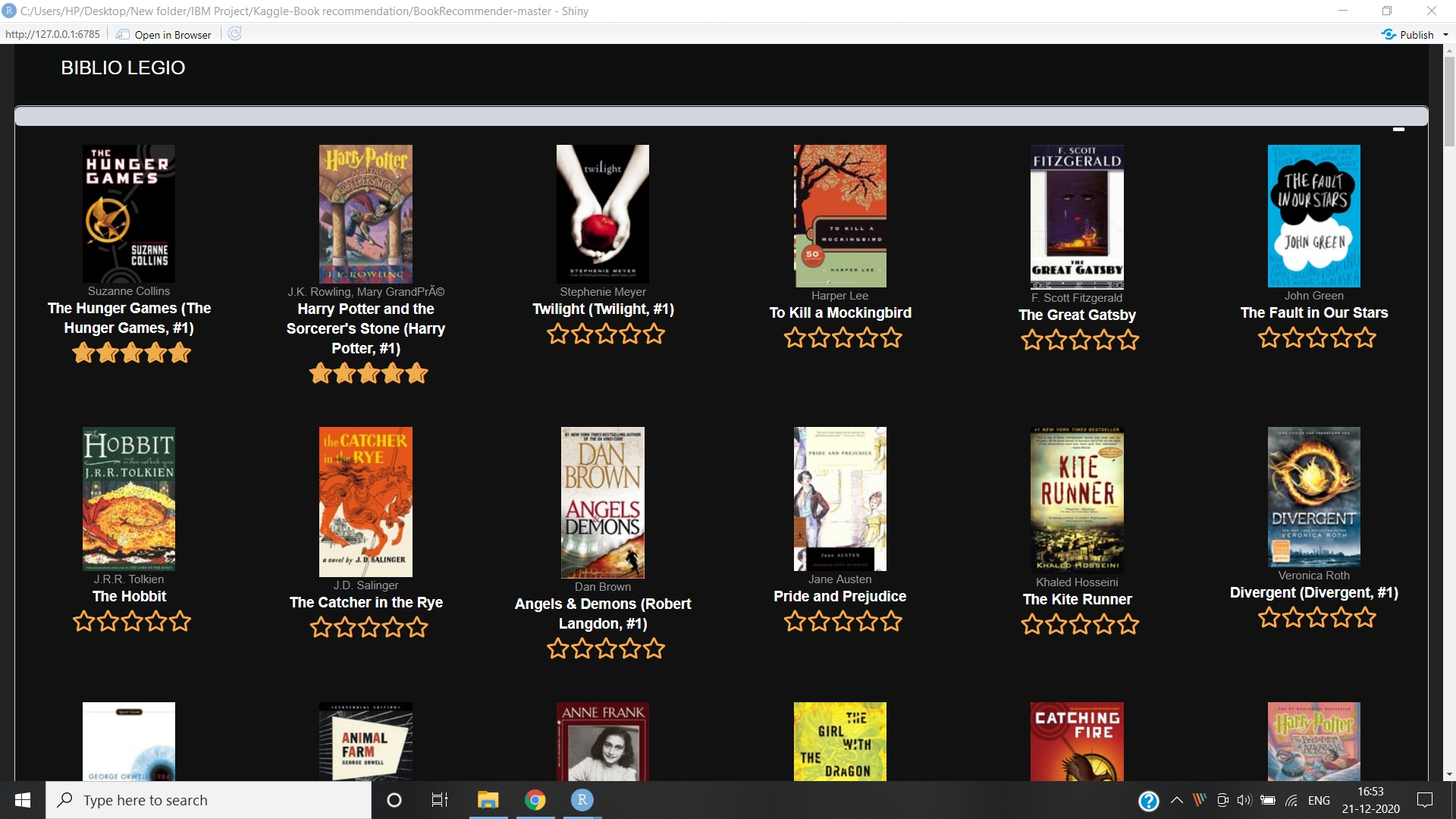
1. loop, method and function is working fine
2. There is no incorrect initialization

**2.4.8 User Acceptance Test**

We did a pilot test with a few users to check whether the book recommender would be acceptable to users. It was found to be very much in tune with the user requirements. And it was giving good results.

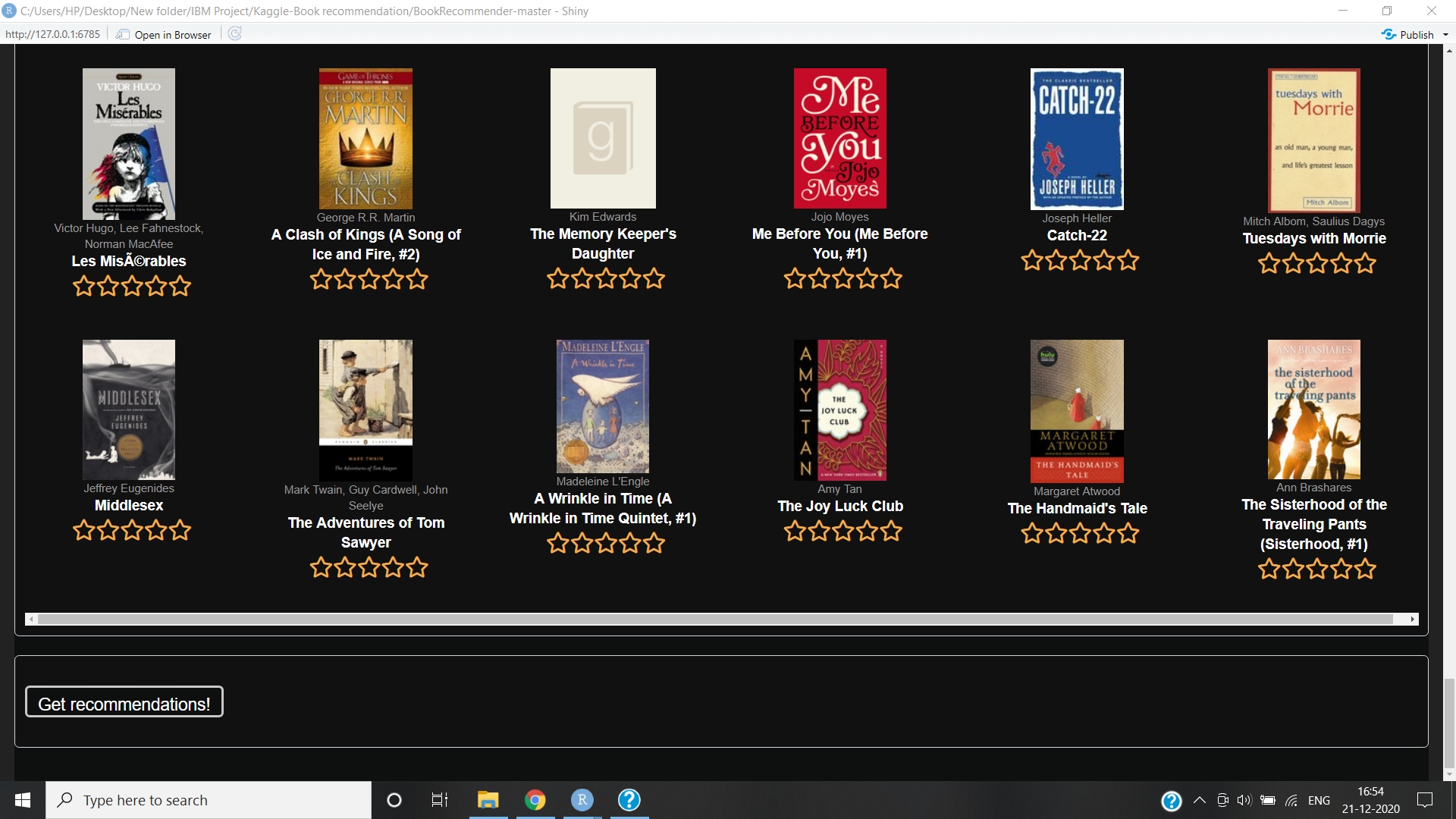
**2.5 Graphical User Interface (GUI) Layout**

**Figure 2**

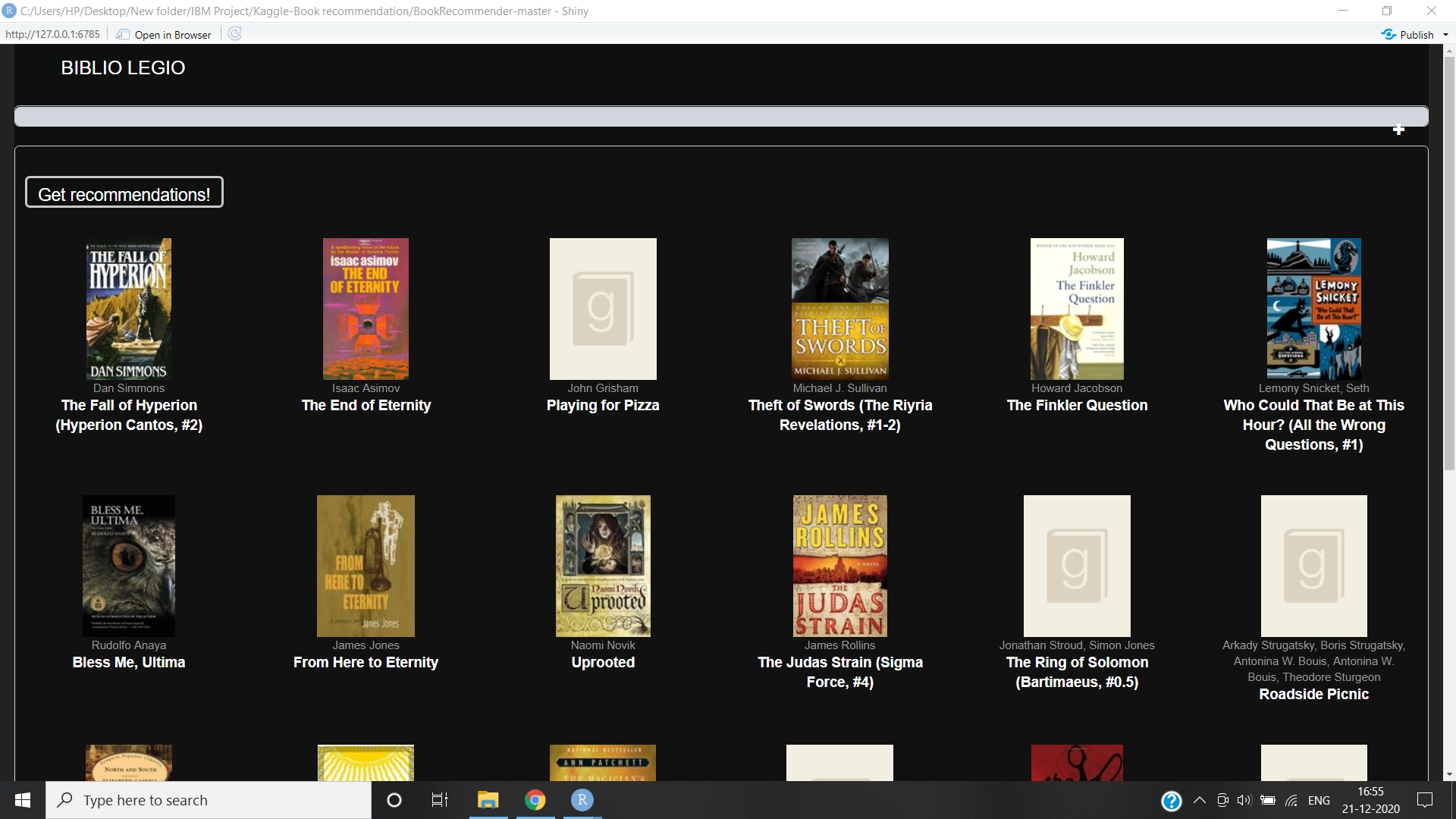


Screenshot for Rating Interface where users can give their ratings.

**Figure 3**

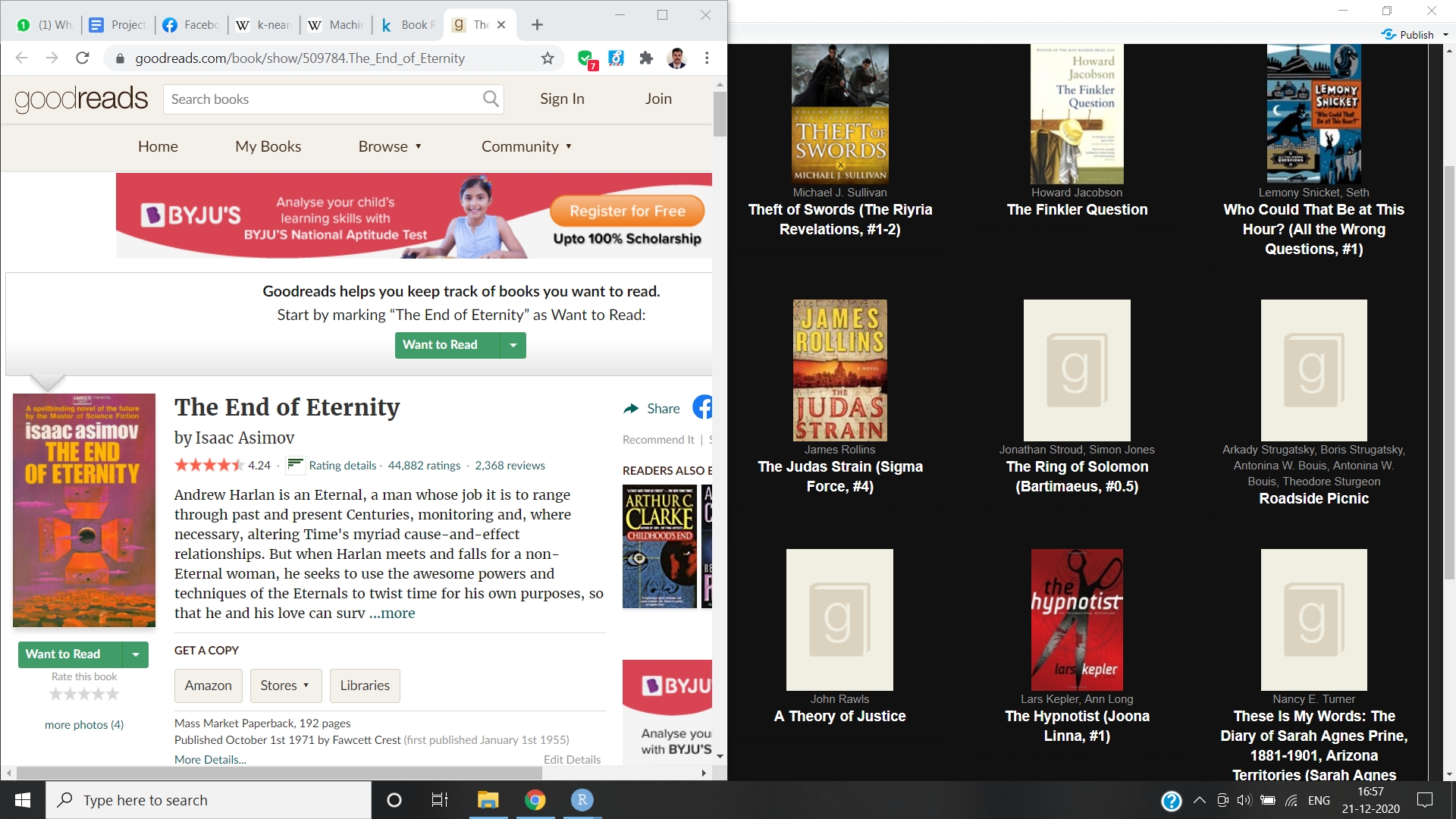
Screenshot for the Get recommendations button

**Figure 4**



Screenshot for book recommendations

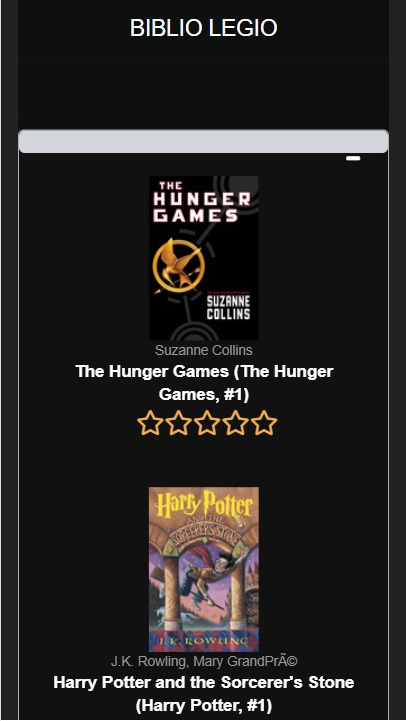
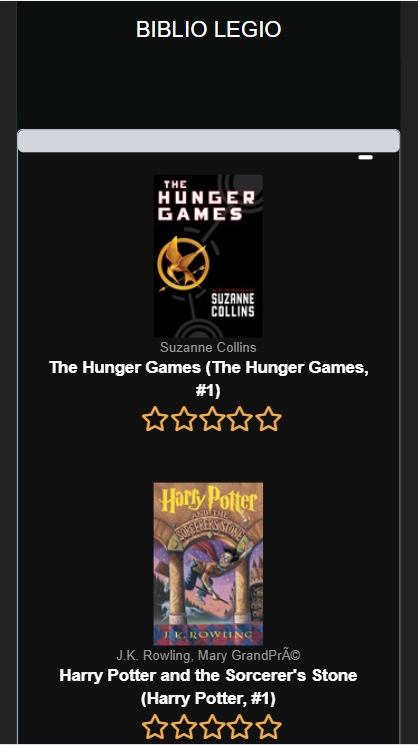
**Figure 5**



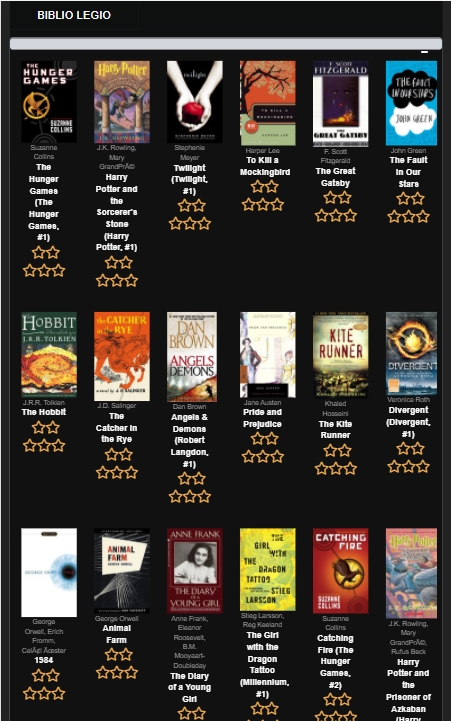
Screenshot for the goodreads book page where user can buy the book

**2.6 Customer testing**

We have checked the responsiveness of the design. The output is compatible for all screen sizes and resolution. This includes PC, Laptop, Tablets, Mobile phones and other handheld devices. Actual testing on different devices could be done once it is hosted on to the cloud through Shinyapps.io. As mentioned in the basic testing by using a sample we have found the design is also user friendly with proper process flow.



Galaxy S5 iPhone 6/7/8



Kindle Fire HDX

**2.7 Evaluation**

We have checked the main functions. The static code was analysed and the entire code evaluated.

**2.7.1 Static Code Analysis**

We used the lintr package of R for doing the static code analysis. Errors other than syntax errors were cleaned up.

**2.7.2 Test of Main Function**

With the test of main function we validated the software system against the functional requirements that we had specified. We tested each function of the application, by providing appropriate input and verifying the output which worked just fine.

**Conclusion**

Readers would like to get recommendations on which are the good books available for them to read. For this they usually talk to people they know who have similar tastes. The Book Recommender system used the same logic to recommend books. This Collaborative filtering system is used to produce efficient and effective large numbers of choices for books, which the user can buy online.Collaborative recommender systems aggregate ratings of objects, recognize commonalities between users on the basis of their ratings, and generate new recommendations. In this project we have used the R programme along with allied packages like Recommender Lab and Shiny to come up with an efficient system.

**Further development or research**

The book recommendation system must recommend books that are of buyer’s interest. Recommendation systems are widely used to recommend products to the end users that are most appropriate. Therefore, we plan to elevate the current system to a user-based system instead of general rating which will result in more personalised recommendation. By implementing a deep learning system to identify the underlying patterns and come up with better recommendations will have superior results.

We can use a Siamese neural network to learn a mapping of product or user features from an item embedding space to a recommendation embedding space. The recommendation embedding of items or users have much lower dimensionality than the NN input and a desired behaviour of being similar as measured by their cosine similarity in the recommendation embedding space. Furthermore, we are working on adding the ISBN database instead of goodreads and also, redirecting users to respective e-commerce sites to buy the recommended book.

**References**

[1] Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, 2011, pp. 1-35

[2] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Bosagh Zadeh WTF: The who-to-follow system at Twitter, Proceedings of the 22nd international conference on World Wide Web

[3]Baran, Remigiusz; Dziech, Andrzej; Zeja, Andrzej "A capable multimedia content discovery platform based on visual content analysis and intelligent data enrichmen[t"](https://doi.org/10.1007/s11042-017-5014-1). Multimedia Tools and Applications. 2018

[4] H. Chen, A. G. Ororbia II, C. L. Giles ExpertSeer: a Keyphrase Based Expert Recommender for Digital Libraries, arXiv preprint, 2015

[5] H. Chen, L. Gou, X. Zhang, C. Giles Collabseer: a search engine for collaboration discovery, in ACM/IEEE Joint Conference on Digital Libraries (JCDL) 2011

[6]Terveen, Loren; Hill, Will "Beyond Recommender Systems: Helping People Help Each Other" (PDF). Addison-Wesley. p. 6. 2001.

[7] An integrated approach to TV & VOD Recommendations Archived 2012, Wayback Machine

[8] Mitchell, Tom Machine Learning. New York: McGraw Hill. .

[9] Koza, John R.; Bennett, Forrest H.; Andre, David; Keane, Martin A. (1996). Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. Artificial Intelligence in Design '96. Springer, Dordrecht. pp. 151–170.

[10] Altman, Naomi S[.](https://en.wikipedia.org/wiki/Naomi_Altman) (1992). ["](https://ecommons.cornell.edu/bitstream/1813/31637/1/BU-1065-MA.pdf)An introduction to kernel and nearest-neighbor nonparametric regression["](https://ecommons.cornell.edu/bitstream/1813/31637/1/BU-1065-MA.pdf) (PDF). The American Statistician. 46 (3): 175–185.

[11]http://fastml.com/goodbooks-10k-a-new-dataset-for-book-recommendations/ [12] http://ieeexplore.ieee.org/document/7435717

[13]https://rstudio-pubs-static.s3.amazonaws.com/506289\_15da1c627ead4b978976526a9bb70b65.html# (using good reads)

[14] Building a recommendation system with R by Suresh K Gorakala and Michele Usuelli

**Appendix**

**#EXPLORATORY ANALYSIS**

library(recommenderlab)

library(data.table)

library(dplyr)

library(tidyr)

library(ggplot2)

library(stringr)

library(DT)

library(knitr)

library(grid)

library(gridExtra)

library(corrplot)

library(qgraph)

library(methods)

library(Matrix)

books <- fread('data/books.csv')

ratings <- fread('data/ratings.csv')

book\_tags <- fread('data/book\_tags.csv')

tags <- fread('data/tags.csv')

#Cleaning the dataset

ratings[, N := .N, .(user\_id, book\_id)]

## corresponding dplyr code

# ratings %>% group\_by(user\_id, book\_id) %>% mutate(n=n())

cat('Number of duplicate ratings: ', nrow(ratings[N > 1]))

#first remove the duplicate ratings

ratings <- ratings[N == 1]

ratings

#Then let's remove users who rated fewer than 3 books

ratings <- ratings[N > 2]

#

set.seed(1)

user\_fraction <- 0.2

users <- unique(ratings$user\_id)

sample\_users <- sample(users, round(user\_fraction \* length(users)))

cat('Number of ratings (before): ', nrow(ratings))

## Number of ratings (before): 960595

ratings <- ratings[user\_id %in% sample\_users]

cat('Number of ratings (after): ', nrow(ratings))

# Exploratory 1

ratings %>%

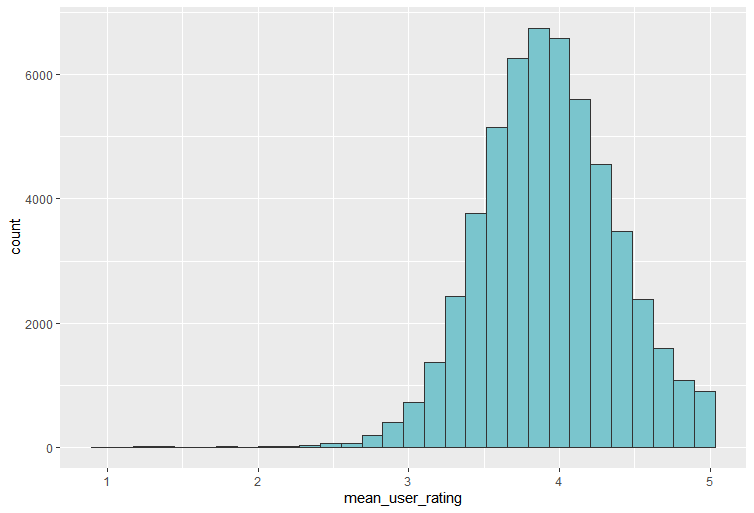
group\_by(user\_id) %>%

summarize(mean\_user\_rating = mean(rating)) %>%

ggplot(aes(mean\_user\_rating)) +

geom\_histogram(fill = "cadetblue3", color = "grey20")

#From this analysis it is clear that the most of the rating are in the range 3-5, with a mean rating of 4.



#Exploratory 2

#By looking at the books that were rated most often we can get an impression of the popularity of a book

books %>%

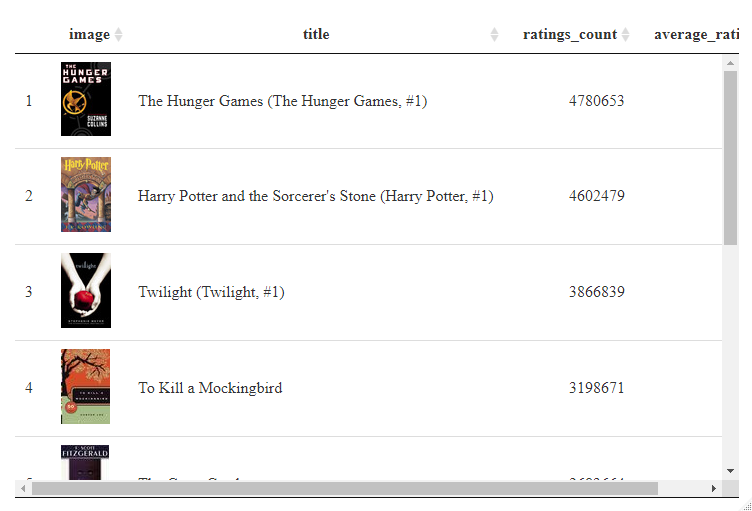
mutate(image = paste0('<img src="', small\_image\_url, '"></img>')) %>%

arrange(-ratings\_count) %>%

top\_n(10,wt = ratings\_count) %>%

select(image, title, ratings\_count, average\_rating) %>%

datatable(class = "nowrap hover row-border", escape = FALSE, options = list(dom = 't',scrollX = TRUE, autoWidth = TRUE))



#Exploratory 3

#Plot the average rating as a function of the length of the title

books <- books %>%

mutate(title\_cleaned = str\_trim(str\_extract(title, '([0-9a-zA-Z]| |\'|,|\\.|\\\*)\*')),

title\_length = str\_count(title\_cleaned, " ") + 1)

tmp <- books %>%

group\_by(title\_length) %>%

summarize(n = n()) %>%

mutate(ind = rank(title\_length))

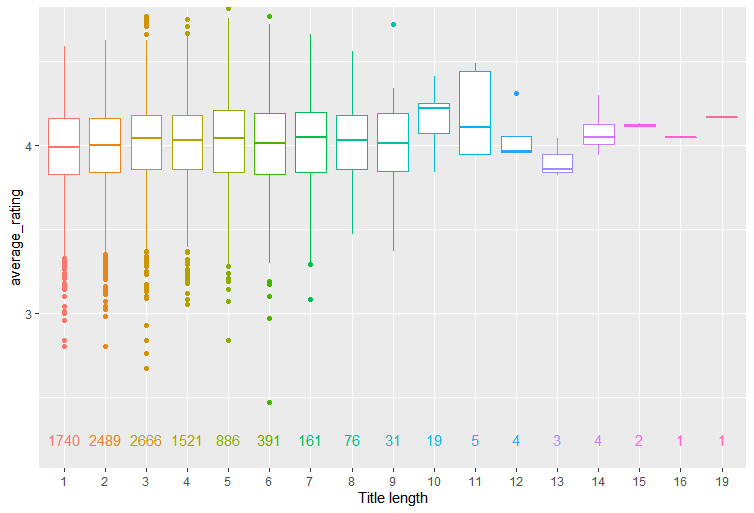
books %>%

ggplot(aes(factor(title\_length), average\_rating, color=factor(title\_length), group=title\_length)) +

geom\_boxplot() + guides(color = FALSE) + labs(x = "Title length") + coord\_cartesian(ylim = c(2.2,4.7)) + geom\_text(aes(x = ind,y = 2.25,label = n), data = tmp)

# We can see that there is in fact some variation in average rating depending on title length.

#Titles with 3 or 7 words seem to have slightly higher ratings.



#Exploratory 4

#The dataset that we used contains some books in different languages

p1 <- books %>%

mutate(language = factor(language\_code)) %>%

group\_by(language) %>%

summarize(number\_of\_books = n()) %>%

arrange(-number\_of\_books) %>%

ggplot(aes(reorder(language, number\_of\_books), number\_of\_books, fill = reorder(language, number\_of\_books))) +

geom\_bar(stat = "identity", color = "grey20", size = 0.35) + coord\_flip() +

labs(x = "language", title = "english included") + guides(fill = FALSE)

p2 <- books %>%

mutate(language = factor(language\_code)) %>%

filter(!language %in% c("en-US", "en-GB", "eng", "en-CA", "")) %>%

group\_by(language) %>%

summarize(number\_of\_books = n()) %>%

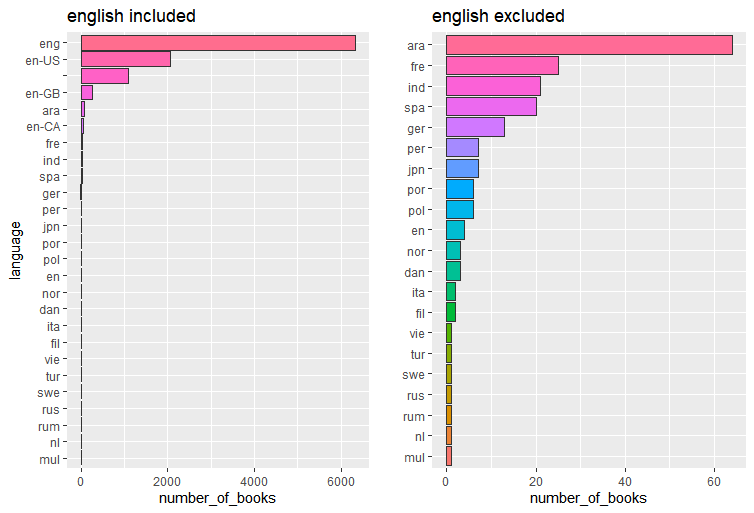
arrange(-number\_of\_books) %>%

ggplot(aes(reorder(language, number\_of\_books), number\_of\_books, fill = reorder(language, number\_of\_books))) +

geom\_bar(stat = "identity", color = "grey20", size = 0.35) + coord\_flip() +

labs(x = "", title = "english excluded") + guides(fill = FALSE)

grid.arrange(p1,p2, ncol=2)



#Exploratory 5

# Plot average rating as a function of boks having subtitle

books <- books %>%

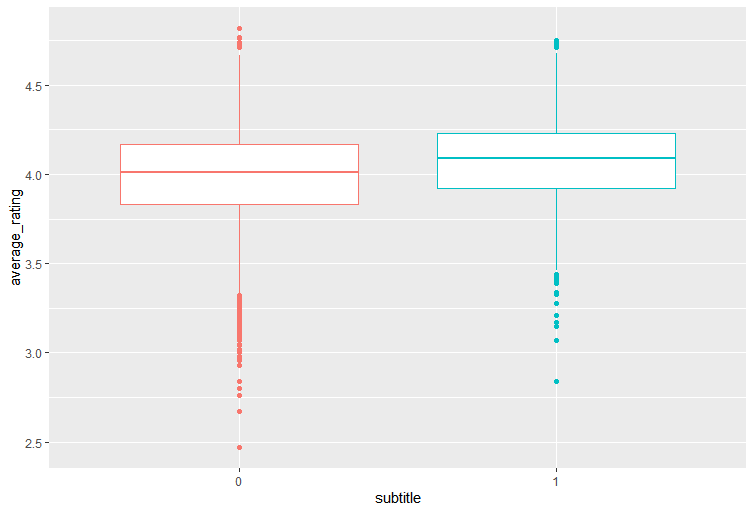
mutate(subtitle = str\_detect(books$title, ':') \* 1, subtitle = factor(subtitle))

books %>%

ggplot(aes(subtitle, average\_rating, group = subtitle, color = subtitle)) +

geom\_boxplot() + guides(color = FALSE)

#We see that books that have a subtitle get rated slightly higher than books without a subtitle.



**#UI CODE**

library(shiny)

#install.packages("shinydashboard")

library(shinydashboard)

#install.packages("recommenderlab")

library(recommenderlab)

library(data.table)

#install.packages("ShinyRatingInput")

#install.packages("digest")

#install.packages("devtools")

library(devtools)

#install.packages("glue")

#updateR

#devtools::install\_github("stefanwilhelm/ShinyRatingInput",force=TRUE)

#install.packages("shinyjs")

#install.packages("shinythemes")

#install.packages("shinyDashboardThemeDIY")

#install.packages("dashboardthemes")

library(ShinyRatingInput)

writeLines('PATH="${RTOOLS40\_HOME}\\usr\\bin;${PATH}"', con = "~/.Renviron")

library(shinyjs)

library(shinythemes)

source('functions/helpers.R')

library(dashboardthemes)

#UI part

shinyUI(

fluidPage(

theme = shinytheme("darkly"),

dashboardPage(

#skin = "yellow",

dashboardHeader(title = "BIBLIO LEGIO"),

dashboardSidebar(disable = TRUE),

dashboardBody(

shinyDashboardThemeDIY(#design

### general

appFontFamily = "Arial"

,appFontColor = "rgb(0,0,0)"

,primaryFontColor = "rgb(0,0,0)"

,infoFontColor = "rgb(0,0,0)"

,successFontColor = "rgb(0,0,0)"

,warningFontColor = "rgb(0,0,0)"

,dangerFontColor = "rgb(0,0,0)"

,bodyBackColor = "rgb(14,15,15)"

### header

,logoBackColor = "rgb(14,15,15)"

,headerButtonBackColor = "rgb(238,238,238)"

,headerButtonIconColor = "rgb(75,75,75)"

,headerButtonBackColorHover = "rgb(210,210,210)"

,headerButtonIconColorHover = "rgb(0,0,0)"

,headerBackColor = "rgb(14,15,15)"

,headerBoxShadowColor = "rgb(14,15,15)"

,headerBoxShadowSize = "2px 2px 2px"

### sidebar

,sidebarBackColor = cssGradientThreeColors(

direction = "down"

,colorStart = "rgb(20,97,117)"

,colorMiddle = "rgb(56,161,187)"

,colorEnd = "rgb(3,22,56)"

,colorStartPos = 0

,colorMiddlePos = 50

,colorEndPos = 100

)

,sidebarPadding = 0

,sidebarMenuBackColor = "transparent"

,sidebarMenuPadding = 0

,sidebarMenuBorderRadius = 0

,sidebarShadowRadius = "0px 0px 0px"

,sidebarShadowColor = "#aaaaaa"

,sidebarUserTextColor = "rgb(255,255,255)"

,sidebarSearchBackColor = "rgb(55,72,80)"

,sidebarSearchIconColor = "rgb(153,153,153)"

,sidebarSearchBorderColor = "rgb(55,72,80)"

,sidebarTabTextColor = "rgb(255,255,255)"

,sidebarTabTextSize = 13

,sidebarTabBorderStyle = "none none solid none"

,sidebarTabBorderColor = "rgb(35,106,135)"

,sidebarTabBorderWidth = 1

,sidebarTabBackColorSelected = cssGradientThreeColors(

direction = "right"

,colorStart = "rgba(44,222,235,1)"

,colorMiddle = "rgba(44,222,235,1)"

,colorEnd = "rgba(0,255,213,1)"

,colorStartPos = 0

,colorMiddlePos = 30

,colorEndPos = 100

)

,sidebarTabTextColorSelected = "rgb(0,0,0)"

,sidebarTabRadiusSelected = "0px 20px 20px 0px"

,sidebarTabBackColorHover = cssGradientThreeColors(

direction = "right"

,colorStart = "rgba(44,222,235,1)"

,colorMiddle = "rgba(44,222,235,1)"

,colorEnd = "rgba(0,255,213,1)"

,colorStartPos = 0

,colorMiddlePos = 30

,colorEndPos = 100

)

,sidebarTabTextColorHover = "rgb(50,50,50)"

,sidebarTabBorderStyleHover = "none none solid none"

,sidebarTabBorderColorHover = "rgb(75,126,151)"

,sidebarTabBorderWidthHover = 1

,sidebarTabRadiusHover = "0px 20px 20px 0px"

### boxes

,boxBackColor = "rgb(255,255,255)"

,boxBorderRadius = 5

,boxShadowSize = "0px 1px 1px"

,boxShadowColor = "rgba(0,0,0,.1)"

,boxTitleSize = 16

,boxDefaultColor = "rgb(210,214,220)"

,boxPrimaryColor = "rgba(44,222,235,1)"

,boxInfoColor = "rgb(210,214,220)"

,boxSuccessColor = "rgb(14,15,15)"

,boxWarningColor = "rgb(244,156,104)"

,boxDangerColor = "rgb(255,88,55)"

,tabBoxTabColor = "rgb(255,255,255)"

,tabBoxTabTextSize = 14

,tabBoxTabTextColor = "rgb(0,0,0)"

,tabBoxTabTextColorSelected = "rgb(0,0,0)"

,tabBoxBackColor = "rgb(255,255,255)"

,tabBoxHighlightColor = "rgba(44,222,235,1)"

,tabBoxBorderRadius = 5

### inputs

,buttonBackColor = "rgb(14,15,15)"

,buttonTextColor = "rgb(0,0,0)"

,buttonBorderColor = "rgb(200,200,200)"

,buttonBorderRadius = 5

,buttonBackColorHover = "rgb(17,18,18)"

,buttonTextColorHover = "rgb(100,100,100)"

,buttonBorderColorHover = "rgb(17,18,18)"

,textboxBackColor = "rgb(255,255,255)"

,textboxBorderColor = "rgb(200,200,200)"

,textboxBorderRadius = 5

,textboxBackColorSelect = "rgb(245,245,245)"

,textboxBorderColorSelect = "rgb(200,200,200)"

### tables

,tableBackColor = "rgb(255,255,255)"

,tableBorderColor = "rgb(240,240,240)"

,tableBorderTopSize = 1

,tableBorderRowSize = 1

),

#shinyDashboardThemes(

#theme = "grey\_dark"),

includeCSS("css/books.css"),

#for the rating box

fluidRow(

#theme = shinytheme("darkly"),

box(background = "black",width = 20, status = "info", solidHeader = TRUE, collapsible = TRUE,

div(class = "rateitems",

uiOutput('ratings')

)

)

),

#for the recommendation box

fluidRow(

theme = shinytheme("darkly"),

useShinyjs(),

box(background = "black",

width = 20, status = "info", solidHeader = TRUE,

br(),

withBusyIndicatorUI(

actionButton(background = "black","btn", "Get recommendations!", class = "btn-warning")

),

br(),

tableOutput("results")

)

)

)

)

)

)

**## SERVER CODE**

source('functions/cf\_algorithm.R')

source('functions/similarity\_measures.R')

# The user ratings gets stored here

get\_user\_ratings <- function(value\_list) {

dat <- data.table(book\_id = sapply(strsplit(names(value\_list), "\_"), function(x) ifelse(length(x) > 1, x[[2]], NA)),rating = unlist(as.character(value\_list)))

dat <- dat[!is.null(rating) & !is.na(book\_id)]

dat[rating == " ", rating := 0]#setting as zero

dat[, ':=' (book\_id = as.numeric(book\_id), rating = as.numeric(rating))]

dat <- dat[rating > 0]

#Ratings are added and a sparse matrix is made

user\_ratings <- sparseMatrix(i = dat$book\_id,

j = rep(1,nrow(dat)),

x = dat$rating,

dims = c(nrow(ratingmat), 1))

}

books <- fread('data/books.csv')

ratings <- fread('data/ratings\_cleaned.csv')

#Creating books and user matrix and taking value other than zero

ratingmat <- sparseMatrix(ratings$book\_id, ratings$user\_id, x=ratings$rating)

ratingmat <- ratingmat[, unique(summary(ratingmat)$j)]

dimnames(ratingmat) <- list(book\_id = as.character(1:10000), user\_id = as.character(sort(unique(ratings$user\_id))))

#ratingmat

#dimnames(ratingmat)

shinyServer(function(input, output, session) {

# Here the books to be rates are displayed

output$ratings <- renderUI({

num\_rows <- 20 #number of rows of books

num\_books <- 6 # books per row

lapply(1:num\_rows, function(i) {

list(fluidRow(lapply(1:num\_books, function(j) {

#Display

list(box(background = "black",width = 2,

div(style = "text-align:center", img(src = books$image\_url[(i - 1) \* num\_books + j], style = "max-height:150")),

div(style = "text-align:center; color: #999999; font-size: 80%", books$authors[(i - 1) \* num\_books + j]),

div(style = "text-align:center", strong(books$title[(i - 1) \* num\_books + j])),

div(style = "text-align:center; font-size: 150%; color: #f0ad4e;", ratingInput(paste0("select\_", books$book\_id[(i - 1) \* num\_books + j]), label = "", dataStop = 5))))

})))

})

})

#Code for button click linked with helpers

df <- eventReactive(input$btn, {

withBusyIndicatorServer("btn", {

useShinyjs()

jsCode <- "document.querySelector('[data-widget=collapse]').click();"

runjs(jsCode)

#Here we take the rating from user

value\_list <- reactiveValuesToList(input)

user\_ratings <- get\_user\_ratings(value\_list)

# Adding to the rating matrix

rmat <- cbind(user\_ratings, ratingmat)

# Getting the ratings of books that has not been predicted

items\_to\_predict <- which(rmat[, 1] == 0)

prediction\_indices <- as.matrix(expand.grid(items\_to\_predict, 1))

# The CF algorithm

res <- predict\_cf(rmat, prediction\_indices, "ubcf", TRUE, cal\_cos, 1000, FALSE, 2000, 1000)

#Display of the books with highest rating first

user\_results <- sort(res[, 1], decreasing = TRUE)[1:24]

user\_predicted\_ids <- as.numeric(names(user\_results))

recom\_results <- data.table(Rank = 1:24,

Book\_id = user\_predicted\_ids,

Author = books$authors[user\_predicted\_ids],

Title = books$title[user\_predicted\_ids],

Predicted\_rating = user\_results)

})

})

# The recommended books display

output$results <- renderUI({

num\_rows <- 4 #number of rows

num\_books <- 6 #number of books per row

recom\_result <- df()

lapply(1:num\_rows, function(i) {

list(

fluidRow(lapply(1:num\_books, function(j) {

box(background = "black",width = 2, status = "success", solidHeader = TRUE,

div(style = "text-align:center",a(href = paste0('https://www.goodreads.com/book/show/', books$best\_book\_id[recom\_result$Book\_id[(i - 1) \* num\_books + j]]),target='blank',img(src = books$image\_url[recom\_result$Book\_id[(i - 1) \* num\_books + j]], height = 150))),

div(style = "text-align:center; color: #999999; font-size: 80%",books$authors[recom\_result$Book\_id[(i - 1) \* num\_books + j]]),

div(style = "text-align:center; font-size: 100%",strong(books$title[recom\_result$Book\_id[(i - 1) \* num\_books + j]])))

})))

})

})

})

**##HELPERS CODE**

# Set up a button to have an animated loading indicator and a checkmark

# for better user experience and to be use with the corresponding `withBusyIndicator` server function

s

withBusyIndicatorUI <- function(button) {

id <- button[['attribs']][['id']]

div(

`data-for-btn` = id,

button,

span(

class = "btn-loading-container",

hidden(

img(src = "ajax-loader-bar.gif", class = "btn-loading-indicator"),

icon("check", class = "btn-done-indicator")

)

),

hidden(

div(class = "btn-err",

div(icon("exclamation-circle"),

tags$b("Error: "),

span(class = "btn-err-msg")

)

)

)

)

}

# Call this function from the server with the button id that is clicked and the

# expression to run when the button is clicked

withBusyIndicatorServer <- function(buttonId, expr) {

# UX stuff: show the "busy" message, hide the other messages, disable the button

loadingEl <- sprintf("[data-for-btn=%s] .btn-loading-indicator", buttonId)

doneEl <- sprintf("[data-for-btn=%s] .btn-done-indicator", buttonId)

errEl <- sprintf("[data-for-btn=%s] .btn-err", buttonId)

shinyjs::disable(buttonId)

shinyjs::show(selector = loadingEl)

shinyjs::hide(selector = doneEl)

shinyjs::hide(selector = errEl)

on.exit({

shinyjs::enable(buttonId)

shinyjs::hide(selector = loadingEl)

})

# running the code when the button is clicked shows an error message if

# an error occurs or a success message if it completes

tryCatch({

value <- expr

shinyjs::show(selector = doneEl)

shinyjs::delay(2000, shinyjs::hide(selector = doneEl, anim = TRUE, animType = "fade",

time = 0.5))

value

}, error = function(err) { errorFunc(err, buttonId) })

}

# When an error happens after a button click, show the error

errorFunc <- function(err, buttonId) {

errEl <- sprintf("[data-for-btn=%s] .btn-err", buttonId)

errElMsg <- sprintf("[data-for-btn=%s] .btn-err-msg", buttonId)

errMessage <- gsub("^ddpcr: (.\*)", "\\1", err$message)

shinyjs::html(html = errMessage, selector = errElMsg)

shinyjs::show(selector = errEl, anim = TRUE, animType = "fade")

}

#loading css components

appCSS <- "

.btn-loading-container {

margin-left: 10px;

font-size: 1.2em;

}

.btn-done-indicator {

color: green;

}

.btn-err {

margin-top: 10px;

color: red;

}

"