CIS 600: Movie Recommendation System
Group: D-Miners

Binder Pal Kaur
bpkaur@syr.edu

Guneev Kaur
gkaur03@syr.edu

Sanjana Kiran
skiran@syr.edu

Siddhartha Roy Nandi
sroynand@syr.edu

Niraj Sitaula
nsitaula@syr.edu

Abstract

The purpose of the project is to build application which can serve as the movie recommender system. In this project we considered building movie recommender system with simplest method possible and compare results with other model. The system built would make movie recommendation to the users based on the previous of score provided by the users. The goal is to recommend movie to the user similar genre movies depending on the good/bad reviews the user gave for the movies.

Introduction

In general, the purpose of recommendation systems (also known as collaborative filtering systems) is to recommend items which a customer is likely to order or pick. These recommendations are based on statistics of a customer's purchase history or previous activities. Most general examples of recommendation systems can be found used by various companies like *Amazon*, *Netflix*, *Youtube*, and many others. With increase in big data, the recommendation system has also been challenging. There are various ways for recommendations system. For the project we choose to build recommendation system for movies based on the techniques we learned from the course.
This project explores simplest approach to build the recommendation system as stated earlier and improves the result by observing and analyzing the outputs. In addition, alternative data mining approach were also explored and to compare the recommendations made by the different model. In addition, comparison is made with the results from the recommendation from our model to the previous built model on the same dataset.

Related Work

Various algorithms can be applied for the recommendation system some of them are *Clustering*, *K-Nearest Neighbor*, *Association Rules*, *Naive Bayes*, and others. Additionally, techniques like *Singular Value Decomposition* (SVD) can also be very useful on handling high dimensional datasets. [2]
In the previous work on same dataset

Novikova [4], developed and evaluated application based on a collaborative filtering recommender (CFR) system along with online app for the recommendation model. However, the author concludes although the model might be able to give stronger recommendation but it is memory-based collaborative filtering, pairwise correlation of every user becomes more difficult and challenging as the user number rises. [4]
In other work by Nakareseisoon [5] on same dataset, *Apriori* algorithm was applied. The author applies a traditional Market Basket Analysis technique to recommend movies. The author concludes using *Apriori* algorithm may not provide a recommendation in a fine-grained user level but still the technique can be used to make movie suggestions. In our work, we use the *Apriori* algorithm to compare with the simplest model of classification.

Methods

The dataset picked for the project is from *MovieLens* [1], a movie recommendation service. The dataset describes 5-star rating that at the time of download contained ratings for 27278 movies with 20000263 ratings. The dataset contains various information whereas we are mainly interested in *ratings.csv* and *movies.csv*. The *ratings.csv* contains *userId*, *movieId*, *rating*, *timestamp* as an attributes whereas, *movies.csv* contains *movieId*, *title*, *genres* as an attributes. In order to know corresponding movie and it's ratings attribute *movieId* is used to combine information from *movies.csv* and *ratings.csv*. In below sections, we explain steps performed for *pre-processing*, *data exploration*, and *data mining* techniques used.

Data Pre-Processing

After loading both the files, some of the basic pre-processing steps were taken to ensure the data is complete. The first thing noticed was due to the large data, the program could slow down whole as all the processing were done in personal laptops with limited resources. It was found that, similar to *DataFrame* structure, there exists *data.table* which possess remarkable feature of using the tables in database and is able to significantly reduce the programming and computing time or processing the movie and rating datasets in R in comparison to *DataFrame*.
So, necessary switch was made from *DataFrame to data.table* whenever possible which helped in saving resources and faster computation. Data sets were loaded using *fread()* function for *data.table* whereas, regular *read.csv* function was used to load into *DataFrame*.
Furthermore, the fields or attributes that were not considered for the processing were removed which we discuss in later data mining techniques. To remove the missing data *na.omit* function was used but it was not any missing attributes for the dataset. On thorough inspection of both the datasets, it was found that any movie can belong to multiple genres and were separated by '| ' . Thus, any movie that belongs to multiple genre must be counted on both so, we break the rows into multiple rows using *csplit()* function.
Thus, this resulted in splitting the various genres by user id into multiple. For e.g. *Toy Story* may have genre as *drama|animation|comedy*, then:

Id	Movie Title	Genres
1	Toy Story	drama|children|comedy

The data is split into multiple records:

Id	Movie Title	Genres
1	Toy Story	drama
1	Toy Story	children
1	Toy Story	comedy

Then, the genre records and movies records were combined to visualize to get an idea on which genre is being mostly rated by the user. The plot for distribution is attached as below in Figure 1:

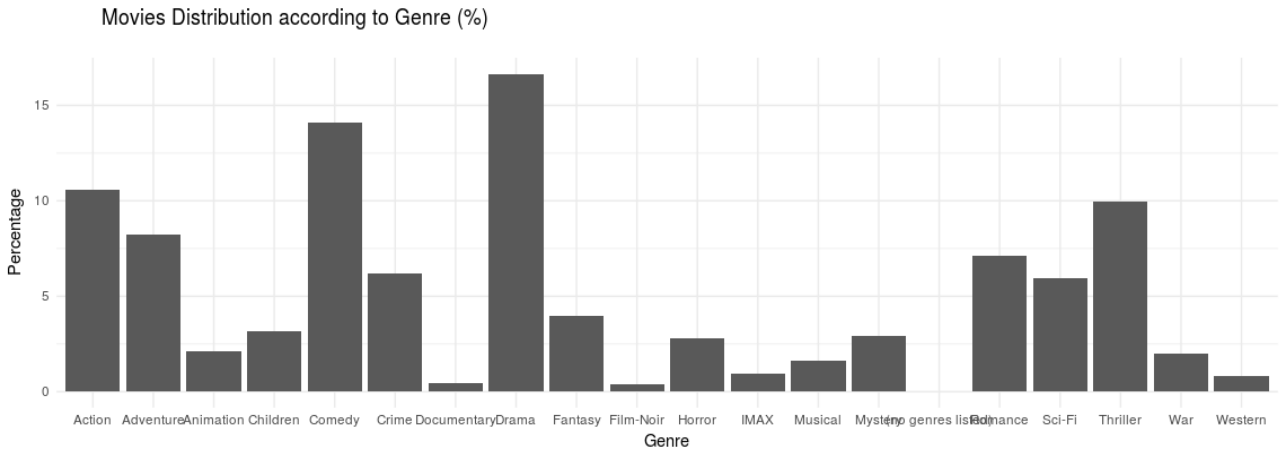

Figure 1: Ratings distributions

In addition, while observing for all unique genres using the *unique* function, 20 genre were found of which one was observed to be missing genre named as 'no genres listed'. There were 246 records missing genre. All the movies with missing genre were removed by neglecting those records.

Data Exploration

For the data exploration we looked into each file individually for the distributions before performing any data mining techniques so that, some prior knowledge can be built by observing the data. The code for building the below graph is attached as Appendix 1.1. First of all, we looked into rating distributions, so we know distribution of number of movies by ratings. Following graph (Figure 2)was observed:

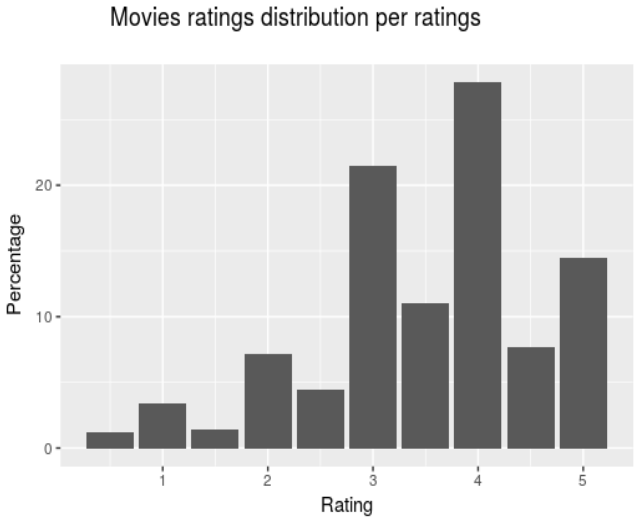

Figure 2: Ratings distributions

From the above figure, we can observe that majority of the movie are above 3 star ratings which can be useful to filter all other movies below 3 star that can be used for the support if necessary to filter low rated movies. For the next *movies* records were explored. As mentioned in earlier section,

after removing missing genre, 19 genres of the movie existed in the dataset. The distribution of each genre of movies in the dataset were explored as shown in the bar graph below in Figure 3:

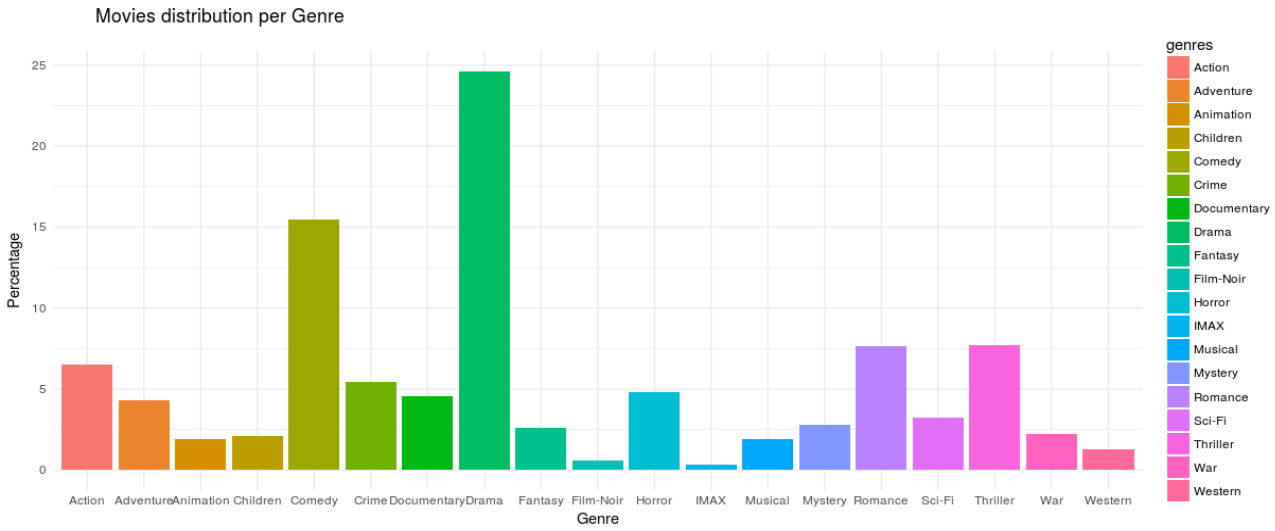

Figure 3: Movies genre distributions

From the above figure, we can observe that dataset is highly dominated by *Drama* genre followed by *Comedy* genre. This shows movies dataset consists mainly of these two genre. This might impact on our classification techniques which is later described.

Data Mining

We used combination of data-mining techniques to form one and an alternative way to build system as well. The implementations Data-mining techniques are explained in following sections with brief introductions:

Classifications with support and Clustering concept:

This is the simplest model we came up with based on the user history. The approach can be further divided into two subparts such that in first portion, we classify or label each user to genre depending on the the majority of the movies of particular genre watched by the user. In addition, we also need to keep track of the list of movies previously watched by the user so that we can ensure, the movie is not repeated in the recommendation. For this purpose, first of all we read the both the files into two *data.table* then, as mentioned in preprocessing step, we split the multiple genre in multiple rows so that we can count each genre user has watched. For e.g. if user watched *Toy Story* which has genre *animation*|*comedy* and same user also watched *Mr. Bean* which belongs to *comedy* genre,then we label user with *comedy* since, it is the majority of the movie watched by the user. However, we apply concept of *Support* by keeping minimum threshold for the user to label or classify to the genre (*in our case we used* 1). Thus, a classification table is created with each user labeled with genres. Since, ratings contains *userId* and

movieId attributes, whereas genre of movies were in *movies* so, both table had to be combined with key of *movieId*. Here, due to large record size use of *data.table* became more useful. In addition, as a preprocessing step for this approach the attributes that were not used were removed i.e. *Title* and *Timestamp* attributes. While one more attribute was added to keep track of the movies watched by the user which is later used for recommending movies. The respective code for this purpose is attached as Appendix 1.2. In addition, if majority of genre user has watched is of multiple then, the one on the top is picked.

For the second part of this method, we use the concept similar of clustering. Clustering refers to the process of grouping a set of physical or abstract objects into classes of similar objects. The clusters has the characteristics of more similarity within the group and are more dissimilar from the objects outside the cluster. In other words, the goal of clustering can be defined as the inter-cluster distance is larger whereas, intra-cluster distance is are as close as possible. It is described as unsupervised learning of a hidden data concept. It identifies groups of related records that can be used as a starting point for exploring further relationships.
However, in our case we can use our prior knowledge of the genres of the movies in the dataset to group the movies of same genre together. Similar to above process, both movies and ratings records were combined such that average rating for each movies could be computed. Next, each movie were separated to corresponding group depending on genre of the movie. The attributes retained for the records were *movieId, genre, Title, and average score.* Thus grouped movies were stored in each separate *DataFrame* such that movies are sorted by average score given by the user to each movies. Hence, for recommendation we can pick top-K movies from the group which are not watched by the user, *where K is the arbitrary value for recommendation.* The code for this purpose is attached as Appendix 1.3.
Now, to recommend the movie for any user, the model expects 4 parameter i.e. *userId, genre or label of user, movie list* and *value of k.* Then, depending on the genre labeled to the user, corresponding group of movies is looked into. Hence, as a result for recommendations, top-k movies from the genre that does not exist the user watched movies list. Although, the *userId* does not have any significance here, the idea behind is if user login application is created then, it would help to keep track of user recommendation. The code for this purpose is attached as Appendix 1.3.

Association Rules:
The next *data mining* approach is *Association Rules* which is similar to the previous work by Nakareseisoon [5]. The main idea behind *association rules* or also known as market basket method, from a set of transaction it finds the rules such that rules can predict the occurrence of one item from given an item based occurrences in the transactions. For e.g. if user watches *movie1* then, how likely is user to watch *movie2.* This rule can be denoted as:
$\{movie1\} \rightarrow \{movie2\}$
Three main concepts attached to the *Association rules* mining are *support, confidence, and lift.* The term support refers to the frequency of an itemset in the transactions i.e. given by:

$$= \frac{count(\{occur.\ of\ itemset\ in\ transactions\})}{\{total\ number\ of\ transactions\}}$$

For the above example, the support for the

rule is calculated as:

$$= \frac{count(\{occur.\ of\ movie1\ , movie2\})}{\{total\ number\ of\ records\ on\ ratings\}}$$

Similarly, *confidence* refers to measure of how often itemset in *right hand side* of the rule occurs given that *left hand side* item occurs i.e. in above example, how often did user watch *movie2* given that they watched *movie1*. For the above example, the confidence for the rule can be calculated as:

$$= \frac{count(\{occur.\ of\ movie1\ , movie2\})}{count(\{occur.\ of\ movie1\ \})}$$

Lift on the other hand is the measure of dependent/correlated events and is computed as:

$$= \frac{support(\{movie1\ , movie2\})}{support\,\{movie1\ \} \times support\,\{movie2\ \}}$$

Furthermore, the additional clause to lift is minimum lift >1 or else the rule found is not considered as a strong rule instead it is misleading. However, the generating rule can be computationally expensive as the itemset grows. Thus, the *Apriori* method can be used to prune the rules. The increment in number of support and confidence can significantly help in pruning the rules. For this approach we use the *arules* package to generate rules. First of all, as in earlier section both *ratings* and *movies* records are merged together. It was noticed, merging larger data records had issue using *DataFrame*, as in earlier section *data.table* seemed to perform well. All the fields except *userId* and *title* which were used to create transactions. The goal of using association rules mining in our project is to given user watched *movieId* which movie can we recommend confidently that also holds the minimum lift value. So, the value for *support*, *confidence*, and the *lift* were set 0.001, 0.9, and 2 respectively as the parameter for the *apriori* function. With maximum length of rule set to 2 it generated 1795 rules. For the next step, the unnecessary braces were removed from the rules and '→ 'was replaced by ', 'such that rules can be split into left hand side (*lhs*) and right hand side (*rhs*). Thus, split *lhs* and *rhs* were stored as record to the column of the *DataFrame* named as *lhs_movie* and *rhs_movie*. For e.g. $\{movie1\} \rightarrow \{movie2\}$ was split such that *lhs_movie* contains *movie*1 , whereas *rhs_movie* contains *movie*2. Hence, for the recommendation, the model expects name of the movie which then uses *filter* function along with *grepl* function to return all the rules that contains input movie in the *lhs_movie* of the rules. The code for this purposen is attached as Appendix 2.1.

Results Evaluation

For the results from each method below is the input and output from each data mining techniques:

Classifications with support and Clustering concept

As mentioned earlier, the model expects 4 parameters. First of all a test user was picked and was given as input to the model suggesting model to recommend 3 movies to the user i.e. k = 3. The output for the user was received as following:

Title	Id	Average Score
Eye In The Sky (Gun chung) (2007)	100743	5.000000
The Old Gun (1975)	127256	5.000000
Bandaged (2009)	129293	5.000000

Table 1: Sample output of model

The list of the movies recommended were not in present in the user movies list however, the movies recommended seemed may be misleading since all were 5.00 star rating. As we will discuss in further sections, the model recommended movies irrespective of number of ratings to the movies. Hence, the result of the recommendation system did not seem reasonable. In further section, we present approach that may be able to handle this scenario to some extent.

Association Rules

For this model, the filter output expects the movie name in left hand side (*lhs*) and depending on the user movie input, all the rules associated to with the movie is displayed. Following example shows the output for the movie named *Cargo* i.e. we are asking model given user had seen *Cargo* what are the other movies that can be recommended that meets the model support, confidence, and lift value.

supp.	conf.	lift	lhs_movie	rhs_movie
0.0010	0.91	15.77	Cargo (2009)	District 9 (2009)
0.0010	0.92	9.10	Cargo (2009)	Inception (2010)
0.0010	0.95	2.57	Cargo (2009)	The (1999)
0.0010	0.90	1.85	Cargo (2009)	Pulp Fiction (1994)

Table 2: Sample output of model

Discussion and Analysis

While first method of using combination methods of classification, grouping, using support seemed initially rationally there are various limitations on it. As mentioned in earlier section, the first method we used was found to be biased towards rating irrespective to the number of ratings i.e. considering a scenario where *movie1* has average rating of 4.5 but rated by 100 users is not picked compared to the *movie2* which has rating of 5.0 but only rated by 1 user. This could be mitigated by adding further *Support* threshold while searching for the movies i.e. checking number of ratings matches minimum threshold. The snippet of code for this purpose for one genre is shown in Appendix 3.1. The sample output obtained after further threshold is shown below:

Title	Id	Average Score
Usual Suspects, The (1995)	50	4.334372
Rear Window (1954)	904	4.271334
Third Man, The (1949)	1212	4.246002

Table 3: Sample output of model

The result obtained seemed more reasonable since $Support = 25$ was applied which ensured that each movie has been rated by atleast 25 different users. Thus, it is better than earlier version of the model. However, there are further limitations, the model is also biased towards majority of the user history of movies. For e.g. if user had watched 10 *action* movies then, initially classification labels user as *Action*. Now, if user attempts to watch *comedy genre* movies, the user would be still be recommended *action genre* movies since, the majority of movie is *action*. So, it does not keep track of user's current choice which is highly considered by current model. If we take example of *Netflix or Youtube*, the recommendation is made in current choices as well.

Similarly *association rules* has it's own limitations. While association rules is computationally expensive, as we increase itemset size the number of rules generated also increase. In addition, as mentioned in earlier work as the author concluded, using *Apriori* algorithm may not provide a recommendation in a fine-grained user level but still the technique can be used to make movie suggestions [5]. As we compare, *association rules* compared to the simplest method, it

seemed more easier to implement the *association rules* since, it is performed using the package. However, both the models are quite different from the point of application. As *association rules* mining does not keep in consideration about the user preference, it is more focused on given any user has watched *movie1* , the possible rules to recommend other movies.
The alternative approach , following the approach in previous work by Berk [8], the sparse matrix can be used for clustering. In this approach, data frames for each genres were created. Using this data frame, further matrix to hold movie/genre was created such that the matrix columns corresponds to genre and the rows values would 0 and 1 if the movie does not belong to the category or belongs to it respectively. For instance, toy story belongs to adventure and animation so, it will have 1 on adventure and animation whereas, 17 columns will be assigned 0. Next, for each user the ratings provided by them were taken such that if the ratings is greater than 3, it was considered as 1, else 0. Hence, the rating matrix was created depending on the method. Furthermore, user profile matrix is calculated by taking dot product of the above two matrices and normalizing the matrix(0/1). Finally, similarity matrix of profile matrix and genres were constructed using Euclidean distance which resulted in of all the users rated movies above 3 that belongs to the particular cluster. So to recommend a movie to user, the model considers the previously rated movies(weighted sum) for the particular user and recommends the next top rated movie in that cluster. [8].
In addition, we also used *Weka* to implement machine learning techniques. However, due to the large data set, we ran out of memory issue. Thus, to overcome this issue we made an attempt to minimize our dataset by only considering few movies from the list. Although, it resolved the issue temporarily, we did not proceed further with the *Weka* since the application of using *R* we were able to handle the data records. Even though it would give brief idea on how to use other data mining technique, however the model build would not be able to perform as efficiently on unseen data.
This also shows the importance of powerful resources in data mining. The data size as we know is growing in very fast paced, and to implement various data mining techniques we need powerful resources that can handle very large data. In our project, we were able to handle the data records but would definitely create problem as the data size grows larger.

Shiny App

Shiny apps helps to share shiny application in web formats and there is a free version available of it. For the project GUI part, we made an attempt to deploy in *shinyapps.io* so that, the model can be accessed using the Internet via link. Although we were successfully able to upload to the web server however, due to the issue on system settings we are not able to access it. As a workaround, we uploaded the code to *Github*. The following two lines of command would help to run the *Shiny* app for the project on the local machine:

```
>> library(shiny)
>> runGitHub("DataMiningProject", "
   bpkaur")
```

Once the above command is run in RStudio given that *shiny* and *rsconnect* packages are installed in the machine, user will be able to see download process of the

folder and the app will be launched. However, the user need to have *movies.csv* and *ratings.csv* in their machine so that it can be uploaded. Although, we were unable to include all the data mining technique for the *Shiny* app portion, the application did help us to implement and understand *Shiny* app. For the future work, we plan to add more features on the *Shiny*. The screenshot for the *Shiny* app is attached in Appendix 5.1.

Conclusion

Hence, we were able to build multiple model for the movie recommendation system. In addition we compared results and analyzed limitation for each of the model. Various data mining techniques learned from the course were successfully applied which helped in getting better understanding of the concepts. For the future work, these model can be further improved by analyzing further on results. Although the model seemed simple but were reasonable with some limitations. In addition, we can also compare these models with current technologies used in the recommendation system to see how each model recommend provided same input.

References

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[2] Rodriguez, Daniel. "Recommender System." Recommender System. N.p., Aug. 2013. Web. 06 May 2017. http://www.cc.uah.es/drg/courses/datamining/IntroRecSys.pdf.

[3] Nakareseisoon, Vitid . "Movie Recommendation with Market Basket Analysis" RPubs - Movie Recommendation with Market Basket Analysis. Web. 06 May 2017. https://rpubs.com/vitidN/203264.

[4] Novikova, Jekaterina. "Building a Movie Recommendation System." Building a Movie Recommendation System. June 2016. Web. 06 May 2017. https://rpubs.com/jeknov/movieRec.

[5] Nakareseisoon, Vitid . "Movie Recommendation with Market Basket Analysis" RPubs - Movie Recommendation with Market Basket Analysis. Web. 06 May 2017. https://rpubs.com/vitidN/203264.

[6] Professor Y. Lin, Lecture notes for CIS 600, Fundamentals of Data and Knowledge Mining, Syracuse University, Spring 2017.

[7] Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. Introduction to data mining. Boston: Pearson Addison Wesley, 2005.

[8] "DATA 643 - Project 1." DATA 643 - Project 1. N.p., 14 Feb. 2017. Web. 07 May 2017. http://api.rpubs.com/HoneyBerk/data643_project1.

Appendix: Appendix1.1 - Code for building ratings distribution graph and genres distribution graph

```
# Build ratings distribution
require('ggplot2')
require(scales)
library(RColorBrewer)

# with faceting

ratings2 = read.csv(rating.fullpath,
                 colClasses = c("NULL","NULL","double","NULL"),
                 sep=",",
                 stringsAsFactors = TRUE)

ggplot(ratings2, aes(x = rating, fill = rating) ) +
  geom_bar( aes(y = ..count..*100/sum(..count..) ) ) +
  xlab("Rating") +
  ylab("Percentage")+
  labs( title = '\tMovies ratings distribution per ratings\n')

unique(movie.table$genres)
```

```
require(data.table)
require('ggplot2')
require(scales)
library(splitstackshape)

movie.fullpath <- "movies.csv"
movie.table <- fread(movie.fullpath)

# Don't care about movie title while merging it just takes more space
movie.table$title <- NULL
movie.table <- cSplit(movie.table, "genres", "|", "long")
movie.table <- movie.table[movie.table$genres != '(no genres listed)', ]

# with faceting
ggplot(movie.table, aes(x = genres, fill = genres) ) +
  geom_bar( aes(y = ..count..*100/sum(..count..) ) ) +
  theme_minimal()+
  xlab("Genre") +
  ylab("Percentage")+
  labs( title = '\tMovies Distribution according to Genre (%)\n')
```

Appendix 1.2: Creating classification table for label or classifying user with genres:

```
# Instead of using data frame we use data.table which is similar and advanced form of
    dataframe
# The purpose here is we will run out of memory using dataframe
# For more information: https://www.analyticsvidhya.com/blog/2016/05/data-table-data-
    frame-work-large-data-sets/
require(data.table)

# This library is used to split column values into different rows
```

```
# If you have not installed you may need to install it
# install.packages("splitstackshape")
library(splitstackshape)

# filepath for movies.csv and ratings.csv
rating.fullpath <- "ratings.csv"
movie.fullpath <- "movies.csv"

# Now the approach can be divided into two parts:
#    1. Classify each user to genre depending on majority of movies of certain genre user
    watched
#       Also track user's watched movie list, it belongs to genre or not so that we do
    not
#       refer user same movie user had watched already
#    2. Group user each genre movies, sort them by average score given by users and these
    movies are picked to
#       recommend each user by comparing top-k movies that do not belong to the user
    movie list

# ****************** Start of PART 1 ************************************************
    ********************
# read movies data and rating data fread is fast or else we run into trouble
rating.table <- fread(rating.fullpath)
movie.table <- fread(movie.fullpath)
rating.table[1,]
# Don't care about movie title while merging it just takes more space
movie.table$title <- NULL
# Don't care about movie title while merging it just takes more space
rating.table$timestamp <- NULL

# now split on the genre to see user's majority likes
movie.table <- cSplit(movie.table, "genres", "|", "long")
movie.table <- movie.table[movie.table$genres != '(no genres listed)', ]

# merge movie and ratings on movieId
# Important point to note is we have split movie genre into multiple rows in movie.table
# So, we need to allow cartesian because movieId one to one mapping will cause lost of
    movies
classification.table <- rating.table[movie.table, on = "movieId", allow.cartesian = TRUE
    ]

# free up some memory since we do not need those tables anymore
remove(rating.table)
remove(movie.table)


# This extra column is track the count of movies in each genre which is later used to
    observe
# the majority of the movies user watched For e.g.
# userId   genre     counter
#   1      drama       1
#   1      action      1
#   1      action      1
```

```
# Now if we sum counter depending on genre we get
# userId   genre     counter
#   1      action       2
#   1      drama        1

# Note: This might be redundant and there may be better way to track it, this is just for
     easiness
# This will take some time
classification.table$counter <- 1

# need to store movie ids for each user to track list of movies user saw in one column
# For all user with specific genre I want to merge to avg score and preserve # number of
    movies
# can be useful to count or threshold
classification.table <- classification.table[, list(avg_score=mean(rating), num_genre_
    movies = sum(counter), movie_list = list(movieId)), by = c('genres','userId')]

# prune less than 2 count, user must have atleast watch 2 movies from category using
    support logic
# this might help to some extent to lower possible records
classification.table <- classification.table[classification.table$num_genre_movies>1,]

# label user with majority of movie genre watched
classification.table <- classification.table[ , .SD[which.max(num_genre_movies)], by = c(
    'userId')]

# Just viewing few contents to see how it looks
classification.table[10, movie_list]
# Now that should satisfy the part 1 requirements
# Note: Need to figure out how to separate training and test data
```

Appendix 1.3: Creating genres and movies to respective genres with average scores:

```
# read movies data and rating data fread is fast or else we run into trouble
rating.table <- fread(rating.fullpath)
movie.table <- fread(movie.fullpath)

# Don't care about movie title while merging it just takes more space
rating.table$timestamp <- NULL
# Don't care about userId as well
rating.table$userId <- NULL

# now split on the genre to pure form
movie.table <- cSplit(movie.table, "genres", "|", "long")
movie.table <- movie.table[movie.table$genres != '(no genres listed)', ]

# merge movie and ratings on movieId
movie.group.table <- rating.table[movie.table, on = "movieId", allow.cartesian = TRUE ]
# free up some memory
remove(rating.table)
remove(movie.table)

# For all movies we need movieId, genre, movieTitle, averageScore
```

```
movie.group.table <- movie.group.table[, list(avg_score=mean(rating)), by = c('genres', '
    movieId', 'title')]

# view all possible genre possible
# unique(movie.group.table$genres)


# These text are created for easiness purpose only since being checked multiple times
text.Adventure <- "Adventure"
text.Animation <- "Animation"
text.Children <- "Children"
text.Comedy <- "Comedy"
text.Fantasy <- "Fantasy"
text.Romance <- "Romance"
text.Drama <- "Drama"
text.Action <- "Action"
text.Crime <- "Crime"
text.Thriller <- "Thriller"
text.Horror <- "Horror"
text.Mystery <- "Mystery"
text.SciFi <- "Sci-Fi"
text.IMAX <- "IMAX"
text.Documentary <- "Documentary"
text.War <- "War"
text.Musical <- "Musical"
text.Western <- "Western"
text.FilmNoir <- "Film-Noir"

# Also removing movies whose score are missing in rating found not all movies have rating
# One way to handle NA value is to remove them using na.omit() function
# There is so much redundant code, try creating function and pass appropiate paramater

# Create dataframe sorted by score for each genre
movie.Adventure <- movie.group.table[movie.group.table$genres == text.Adventure,]
movie.Adventure <- na.omit(movie.Adventure)
movie.Adventure <- movie.Adventure[order(movie.Adventure$avg_score, decreasing = TRUE),]
movie.Adventure <- as.data.frame.matrix(movie.Adventure)

# Create dataframe sorted by score for each genre
movie.Animation <- movie.group.table[movie.group.table$genres == text.Animation,]
movie.Animation <- na.omit(movie.Animation)
movie.Animation <- movie.Animation[order(movie.Animation$avg_score, decreasing = TRUE),]
movie.Animation <- as.data.frame.matrix(movie.Animation)

# Create dataframe sorted by score for each genre
movie.Children <- movie.group.table[movie.group.table$genres == text.Children,]
movie.Children <- na.omit(movie.Children)
movie.Children <- movie.Children[order(movie.Children$avg_score, decreasing = TRUE),]
movie.Children <- as.data.frame.matrix(movie.Children)

# Create dataframe sorted by score for each genre
movie.Comedy <- movie.group.table[movie.group.table$genres == text.Comedy,]
movie.Comedy <- na.omit(movie.Comedy)
movie.Comedy <- movie.Comedy[order(movie.Comedy$avg_score, decreasing = TRUE),]
```

```
movie.Comedy <- as.data.frame.matrix(movie.Comedy)

# Create dataframe sorted by score for each genre
movie.Fantasy <- movie.group.table[movie.group.table$genres == text.Fantasy,]
movie.Fantasy <- na.omit(movie.Fantasy)
movie.Fantasy <- movie.Fantasy[order(movie.Fantasy$avg_score, decreasing = TRUE),]
movie.Fantasy <- as.data.frame.matrix(movie.Fantasy)

# Create dataframe sorted by score for each genre
movie.Romance <- movie.group.table[movie.group.table$genres == text.Romance,]
movie.Romance <- na.omit(movie.Romance)
movie.Romance <- movie.Romance[order(movie.Romance$avg_score, decreasing = TRUE),]
movie.Romance <- as.data.frame.matrix(movie.Romance)

# Create dataframe sorted by score for each genre
movie.Drama <- movie.group.table[movie.group.table$genres == text.Drama,]
movie.Drama <- na.omit(movie.Drama)
movie.Drama <- movie.Drama[order(movie.Drama$avg_score, decreasing = TRUE),]
movie.Drama <- as.data.frame.matrix(movie.Drama)

# Create dataframe sorted by score for each genre
movie.Action <- movie.group.table[movie.group.table$genres == text.Action,]
movie.Action <- na.omit(movie.Action)
movie.Action <- movie.Action[order(movie.Action$avg_score, decreasing = TRUE),]
movie.Action <- as.data.frame.matrix(movie.Action)

# Create dataframe sorted by score for each genre
movie.Crime <- movie.group.table[movie.group.table$genres == text.Crime,]
movie.Crime <- na.omit(movie.Crime)
movie.Crime <- movie.Crime[order(movie.Crime$avg_score, decreasing = TRUE),]
movie.Crime <- as.data.frame.matrix(movie.Crime)

# Create dataframe sorted by score for each genre
movie.Thriller <- movie.group.table[movie.group.table$genres == text.Thriller,]
movie.Thriller <- na.omit(movie.Thriller)
movie.Thriller <- movie.Thriller[order(movie.Thriller$avg_score, decreasing = TRUE),]
movie.Thriller <- as.data.frame.matrix(movie.Thriller)

# Create dataframe sorted by score for each genre
movie.Horror <- movie.group.table[movie.group.table$genres == text.Horror,]
movie.Horror <- na.omit(movie.Horror)
movie.Horror <- movie.Horror[order(movie.Horror$avg_score, decreasing = TRUE),]
movie.Horror <- as.data.frame.matrix(movie.Horror)

# Create dataframe sorted by score for each genre
movie.Mystery <- movie.group.table[movie.group.table$genres == text.Mystery,]
movie.Mystery <- na.omit(movie.Mystery)
movie.Mystery <- movie.Mystery[order(movie.Mystery$avg_score, decreasing = TRUE),]
movie.Mystery <- as.data.frame.matrix(movie.Mystery)

# Create dataframe sorted by score for each genre
movie.SciFi <- movie.group.table[movie.group.table$genres == text.SciFi,]
movie.SciFi <- na.omit(movie.SciFi)
movie.SciFi <- movie.SciFi[order(movie.SciFi$avg_score, decreasing = TRUE),]
```

```
movie.SciFi <- as.data.frame.matrix(movie.SciFi)

# Create dataframe sorted by score for each genre
movie.IMAX <- movie.group.table[movie.group.table$genres == text.IMAX,]
movie.IMAX <- na.omit(movie.IMAX)
movie.IMAX <- movie.IMAX[order(movie.IMAX$avg_score, decreasing = TRUE),]
movie.IMAX <- as.data.frame.matrix(movie.IMAX)

# Create dataframe sorted by score for each genre
movie.Documentary <- movie.group.table[movie.group.table$genres == text.Documentary,]
movie.Documentary <- na.omit(movie.Documentary)
movie.Documentary <- movie.Documentary[order(movie.Documentary$avg_score, decreasing =
    TRUE),]
movie.Documentary <- as.data.frame.matrix(movie.Documentary)

# Create dataframe sorted by score for each genre
movie.War <- movie.group.table[movie.group.table$genres == text.War,]
movie.War <- na.omit(movie.War)
movie.War <- movie.War[order(movie.War$avg_score, decreasing = TRUE),]
movie.War <- as.data.frame.matrix(movie.War)

# Create dataframe sorted by score for each genre
movie.Musical <- movie.group.table[movie.group.table$genres == text.Musical,]
movie.Musical <- na.omit(movie.Musical)
movie.Musical <- movie.Musical[order(movie.Musical$avg_score, decreasing = TRUE),]
movie.Musical <- as.data.frame.matrix(movie.Musical)

# Create dataframe sorted by score for each genre
movie.Western <- movie.group.table[movie.group.table$genres == text.Western,]
movie.Western <- na.omit(movie.Western)
movie.Western <- movie.Western[order(movie.Western$avg_score, decreasing = TRUE),]
movie.Western <- as.data.frame.matrix(movie.Western)

# Create dataframe sorted by score for each genre
movie.FilmNoir <- movie.group.table[movie.group.table$genres == text.FilmNoir,]
movie.FilmNoir <- na.omit(movie.FilmNoir)
movie.FilmNoir <- movie.FilmNoir[order(movie.FilmNoir$avg_score, decreasing = TRUE),]
movie.FilmNoir <- as.data.frame.matrix(movie.FilmNoir)
```

Appendix 1.4: Recommending user movies

```
test_user <- classification.table[10,c("userId", "genres", "movie_list")]
user.genre.label <- test_user$genres
k <- 2
# Using if else loop depending on genre user has been labelled we need to check
    respective dataframe

recommendations <- data.frame(Title=character(),MovieId=numeric(),Score=double())

if (user.genre.label == text.Adventure)
{

  for (id in movie.Adventure$movieId)
  {
```

```
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Adventure [movie.Adventure$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Animation) {
  for (id in movie.Animation$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Animation[movie.Animation$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }

} else if (user.genre.label == text.Children) {

  for (id in movie.Children$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Children[movie.Adventure$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }

} else if (user.genre.label == text.Comedy) {
  for (id in movie.Comedy$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Comedy[movie.Comedy$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break

  }
} else if (user.genre.label == text.Fantasy) {
  for (id in movie.Fantasy$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
```

```
    record <- movie.Fantasy[movie.Fantasy$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
  }
  if (nrow(recommendations) >= k) break

 }
} else if (user.genre.label == text.Romance) {
 for (id in movie.Romance$movieId)
 {
  if (!(id %in% test_user$movie_list))
  {
    record <- movie.Romance[movie.Romance$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
  }
  if (nrow(recommendations) >= k) break

 }

} else if (user.genre.label == text.Drama) {
 for (id in movie.Drama$movieId)
 {
  if (!(id %in% test_user$movie_list))
  {
    record <- movie.Drama[movie.Drama$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
  }
  if (nrow(recommendations) >= k) break

 }
} else if (user.genre.label == text.Action) {
 for (id in movie.Action$movieId)
 {
  if (!(id %in% test_user$movie_list))
  {
    record <- movie.Action[movie.Action$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
  }
  if (nrow(recommendations) >= k) break

 }

} else if (user.genre.label == text.Crime) {
 for (id in movie.Crime$movieId)
 {
  if (!(id %in% test_user$movie_list))
  {
```

```
    record <- movie.Crime[movie.Crime$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
   }
   if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Thriller) {
  # Code for Thriller genre
  for (id in movie.Thriller$movieId)
  {
   if (!(id %in% test_user$movie_list))
   {
    record <- movie.Thriller[movie.Thriller$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score = record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
   }
   if (nrow(recommendations) >= k) break
  }

} else if (user.genre.label == text.Horror) {
  for (id in movie.Horror$movieId)
  {
   if (!(id %in% test_user$movie_list))
   {
    record <- movie.Horror[movie.Horror$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
   }
   if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Mystery) {
  for (id in movie.Mystery$movieId)
  {
   if (!(id %in% test_user$movie_list))
   {
    record <- movie.Mystery[movie.Mystery$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
   }
   if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.SciFi) {
  for (id in movie.SciFi$movieId)
  {
   if (!(id %in% test_user$movie_list))
   {
    record <- movie.SciFi[movie.SciFi$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
```

```
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.IMAX) {
  for (id in movie.IMAX$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.IMAX[movie.IMAX$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Documentary) {
  for (id in movie.Documentary$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Documentary[movie.Documentary$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.War) {
  for (id in movie.War$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.War[movie.War$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Musical){
  for (id in movie.Musical$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Musical[movie.Musical$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Western){
  for (id in movie.Western$movieId)
  {
```

```
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Western[movie.Western$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
}else if (user.genre.label == text.FilmNoir){
  for (id in movie.FilmNoir$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.FilmNoir[movie.FilmNoir$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
}else{
  print("Unknown Genre or no Genre")
}
recommendations
```

Appendix 1.4: Recommending user movies

```
if (user.genre.label == text.Adventure)
{
  for (id in movie.Adventure$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      if (nrow(rating.table[rating.table$movieId == id,]) > 25){
        record <- movie.Adventure [movie.Adventure$movieId == id, ]
        newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record
            $avg_score)
        recommendations <- rbind(recommendations, newRow)
      }
    }
    if (nrow(recommendations) >= k) break
  }
}
```

Appendix 2.1 Code for Association rules:

```
require(data.table)
library(arules)
library(dplyr)
library(stringr)

rm(list=ls())
library(splitstackshape)
```

```
rating.fullpath <- "ratings.csv"
ratings = read.csv(rating.fullpath,
                   colClasses = c("integer","integer","NULL","NULL"),
                   sep=",",
                   stringsAsFactors = FALSE)
ratings <- na.omit(ratings)
rating.table <- data.table(ratings, key = "movieId")
remove(ratings)

movie.fullpath <- "movies.csv"
movie.table <- read.csv(movie.fullpath, sep=",",colClasses = c("integer","character","
    NULL"))
movie.table <- data.table(movie.table, key = "movieId")

ratings.merged <- merge(x = rating.table, y = movie.table, by= "movieId", all.x=TRUE)
rating.df <- as.data.frame(ratings.merged)

rating.df$movieId <- NULL
head(rating.df)

movie_transactions <- as(split(rating.df[,"title"], rating.df[,"userId"]), "transactions"
    )

rules <- apriori(movie_transactions, parameter = list(support = 0.001,
                            confidence = 0.9, maxlen=2))
assoc_rules = as(rules,"data.frame")

head(assoc_rules)

rules = as.character(assoc_rules$rules)
rules = gsub("=>",",",rules)
rules = gsub("\\{","",rules)
rules = gsub("\\}","",rules)
head(rules)

rules = str_split(rules,",")
assoc_rules$lhs_movie = sapply( rules, head, n = 1)
assoc_rules$rhs_movie = sapply( rules , tail, n = 1)
head(assoc_rules)
assoc_rules$rules = NULL

# http://stackoverflow.com/a/24821141/5916727
filter(assoc_rules, grepl('Cargo', lhs_movie))
```

Appendix 5.1 Shiny App Uploading both the files and displaying the contents. 25 items at a time which can be changed dynamically by the user to any number. After uploading both the files you can switch on the tabs and see any of the files.

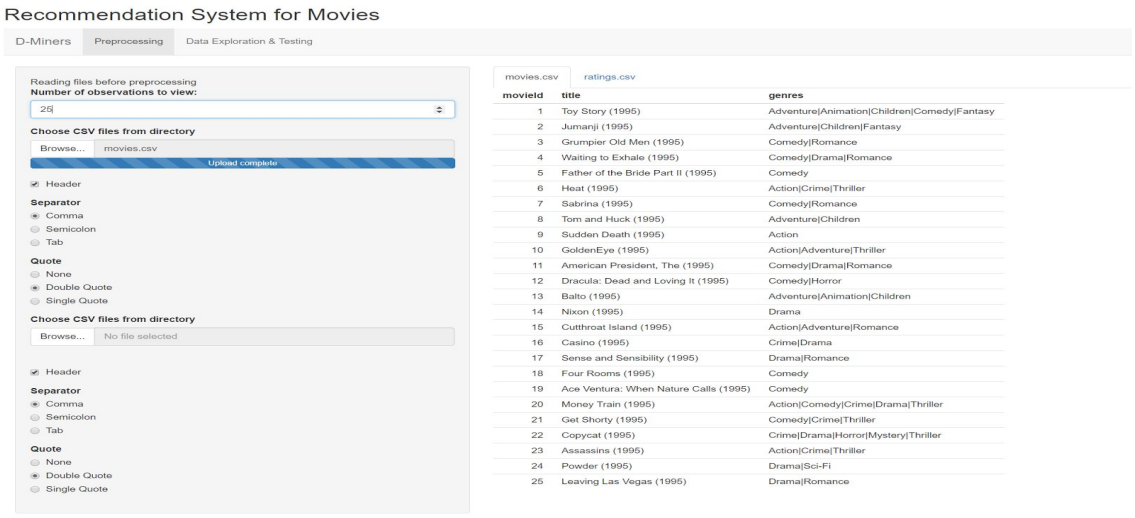

Figure 4: Shiny App 1

Next part is displaying the contents stepwise and displaying test results First we split the movies.csv file for unique genre and then we merged it with the ratings.csv and added a counter column to it so that we can keep track how many movies used has watched. Then we performed aggregation using user Id and genre and calculated average score, which is as shown below in the screenshot

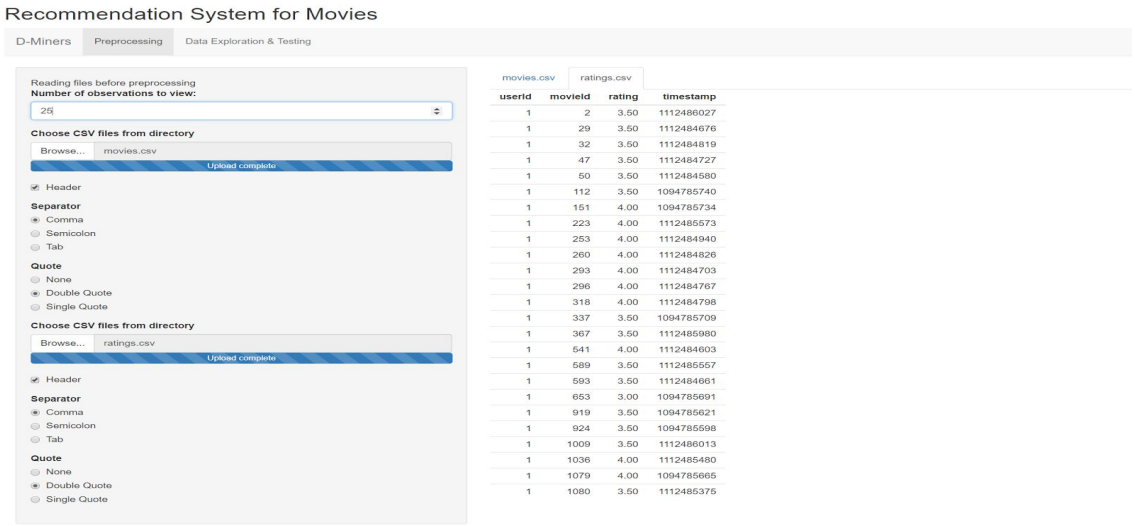

Figure 5: Shiny App 2

After this, we pruned the dataset and only kept the entries where each user has watched minimum two movies, and we then calculated average score for each movie according to their genre. One movie can be there in more than one genre. It is as shown in the screenshot below:

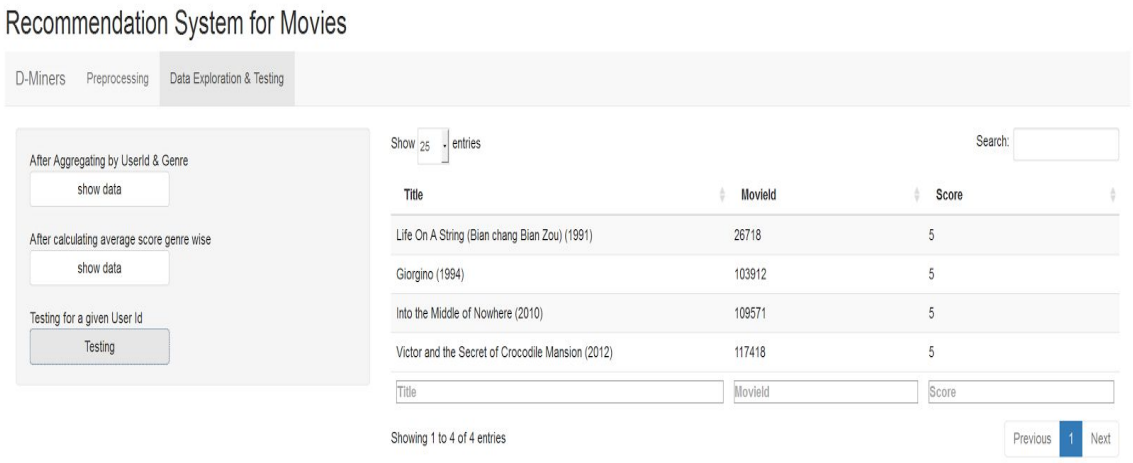

Figure 6: Shiny App 3

Now that we have all the movies sorted according to their genre and average score, we performed simple testing by choosing an existing user. We are checking which genre user belongs to and then we are recommending top K movies to the user which he has not watched yet. Which is shown in the screenshot below:

Code: This file is the designing of user interface for shiny app

```
# ui.R
library(shiny)
if (interactive()){

shinyUI(fluidPage(
  titlePanel("Recommendation System for Movies"),
  navbarPage("D-Miners",
    tabPanel("Preprocessing",
            sidebarLayout( position = "left",
                        sidebarPanel(
                           "Reading files before preprocessing",
                          numericInput("obs", "Number of observations to view:", 25),
                          fileInput("movies",
                                    "Choose CSV files from directory",
                                    multiple = TRUE,
                                    accept=c('text/csv',
                                             'text/comma-separated-values,text/plain',
                                             '.csv')),
                          checkboxInput('header1', 'Header', TRUE),
                          radioButtons('sep1', 'Separator',
                                       c(Comma=',',
                                         Semicolon=';',
                                         Tab='\t'),
                                       ','),
                          radioButtons('quote1', 'Quote',
                                       c(None='',
                                         'Double Quote'='"',
                                         'Single Quote'="'"),
```

```
                                '"'),
                    fileInput("ratings",
                              "Choose CSV files from directory",
                              multiple = TRUE,
                              accept=c('text/csv',
                                       'text/comma-separated-values,text/plain',
                                       '.csv')),
                    checkboxInput('header2', 'Header', TRUE),
                    radioButtons('sep2', 'Separator',
                                 c(Comma=',',
                                   Semicolon=';',
                                   Tab='\t'),
                                 ','),
                    radioButtons('quote2', 'Quote',
                                 c(None='',
                                   'Double Quote'='"',
                                   'Single Quote'="'"),
                                 '"')
                  ),
                  mainPanel(
                    tabsetPanel(
                      tabPanel("movies.csv", tableOutput("contents1")),
                      tabPanel("ratings.csv", tableOutput("contents2"))
                              )
                         )
                  )
),


        tabPanel("Data Exploration & Testing",
                sidebarLayout( position = "left",
                             sidebarPanel(
                               "After Aggregating by UserId & Genre",
                                br(),
                                         actionButton("afteragg", "show data", width
                                             = 200),
                                         br(), br(),
                               "After calculating average score genre wise",
                               br(),
                                         actionButton("aftermerge", "show data",
                                             width = 200),
                                         br(), br(),
                               "Testing for a given User Id",
                               br(),
                                         actionButton("testing", "Testing", width =
                                             200)

                                         ),
                             mainPanel(
                               fluidRow(
                                 column(
                                   dataTableOutput('table'), width=12)
                                    )
                                   )
```

```
                                    )
                    )
                    )
            ))

}
```

server.R

```
library(shiny)
#changing the max size so that we will be able to upload large files
#which exceeds max size like ratings.csv in our project
options(shiny.maxRequestSize = 5000*1024^2)
require(data.table)

# This library is used to split column values into different rows
# If you have not installed you may need to install it
# install.packages("splitstackshape")
library(splitstackshape)
# filepath for movies.csv and ratings.csv
rating.fullpath <- "ratings.csv"
movie.fullpath <- "movies.csv"
# Now the approach can be divided into two parts:
#     1. Classify each user to genre depending on majority of movies of certain genre user
     watched
#        Also track user's watched movie list, it belongs to genre or not so that we do
    not
#        refer user same movie user had watched already
#     2. Group user each genre movies, sort them by average score given by users and these
     movies are picked to
#        recommend each user by comparing top-k movies that do not belong to the user
    movie list

shinyServer(function(input, output) {
  # to display the contents of movies.csv
  output$contents1 <- renderTable({
    # input$file1 will be NULL initially. After the user selects
    # and uploads a file, it will be a data frame with 'name',
    # 'size', 'type', and 'datapath' columns. The 'datapath'
    # column will contain the local filenames where the data can
    # be found.
    inFile <- input$movies
    if (is.null(inFile))
      return(NULL)
  head(fread(movie.fullpath), n = input$obs)
  })

  # to display the contents of ratings.csv
  output$contents2 <- renderTable({

    inFile <- input$ratings
    if (is.null(inFile))
      return(NULL)
    head(fread(rating.fullpath), n = input$obs)
```

```
  })

# ******************Start of PART 1 ***************************************************
# read movies data and rating data fread is fast or else we run into trouble
  rating.table <- fread(rating.fullpath)
  movie.table <- fread(movie.fullpath)
# Don't care about movie title while merging it just takes more space
movie.table$title <- NULL
# Don't care about movie title while merging it just takes more space
rating.table$timestamp <- NULL

# now split on the genre to see user's majority likes
movie.table <- cSplit(movie.table, "genres", "|", "long")
movie.table <- movie.table[movie.table$genres != '(no genres listed)', ]

# merge movie and ratings on movieId
# Important point to note is we have split movie genre into multiple rows in movie.table
# So, we need to allow cartesian because movieId one to one mapping will cause lost of
    movies
classification.table <- rating.table[movie.table, on = "movieId", allow.cartesian = TRUE
    ]
classification.table1 <- head(classification.table,100)
#displaying the data after aggregation
observeEvent(input$afteragg, {

  output$table <- renderDataTable(
    classification.table1)
})

# free up some memory since we do not need those tables anymore
remove(rating.table)
remove(movie.table)
remove(classification.table)
# This extra column is track the count of movies in each genre which is later used to
    observe
# the majority of the movies user watched For e.g.
# userId   genre     counter
#   1      drama       1
#   1      action      1
#   1      action      1
# Now if we sum counter depending on genre we get
# userId   genre     counter
#   1      action       2
#   1      drama        1
# Note: This might be redundant and there may be better way to track it, this is just for
     easiness
# This will take some time
classification.table1$counter <- 1
# need to store movie ids for each user to track list of movies user saw in one column
# For all user with specific genre I want to merge to avg score and preserve # number of
    movies
# can be useful to count or threshold
classification.table1 <- classification.table1[, list(avg_score=mean(rating), num_genre_
```

```
    movies = sum(counter), movie_list = list(movieId)), by = c('genres','userId')]

# prune less than 2 count, user must have atleast watch 2 movies from category using
    support logic
# this might help to some extent to lower possible records
#classification.table1 <- classification.table1[classification.table1$num_genre_movies
    >1,]

# label user with majority of movie genre watched
classification.table1 <- classification.table1[ , .SD[which.max(num_genre_movies)], by =
    c('userId')]
# Just viewing few contents to see how it looks
#classification.table[10, movie_list]
# Now that should satisfy the part 1 requirements
# Note: Need to figure out how to separate training and test data

# ------------------ End of PART 1
    ------------------------------------------------------------------------


# ****************** Start of PART 2 ****************************************************
    *******

# Quick Reminder: # Group movies in 19 different genre and sort them in average score
    decreasing order
#                 this decreasing score is used to recommend movies for the user
#                 use k movies that does not exist in user list
# read movies data and rating data fread is fast or else we run into trouble
rating.table <- fread(rating.fullpath)
movie.table <- fread(movie.fullpath)

# Don't care about movie title while merging it just takes more space
rating.table$timestamp <- NULL
# Don't care about userId as well
rating.table$userId <- NULL

# now split on the genre to pure form
movie.table <- cSplit(movie.table, "genres", "|", "long")
movie.table <- movie.table[movie.table$genres != '(no genres listed)', ]

# merge movie and ratings on movieId
movie.group.table <- rating.table[movie.table, on = "movieId", allow.cartesian = TRUE ]


#displaying the data after calculating average score
observeEvent(input$aftermerge, {

  output$table <- renderDataTable(
    movie.group.table)
})

# free up some memory
remove(rating.table)
remove(movie.table)
```

```
# For all movies we need movieId, genre, movieTitle, averageScore
movie.group.table <- movie.group.table[, list(avg_score=mean(rating)), by = c('genres', '
    movieId', 'title')]

# view all possible genre possible
# unique(movie.group.table$genres)
# Adventure
# Animation
# Children
# Comedy
# Fantasy
# Romance
# Drama
# Action
# Crime
# Thriller
# Horror
# Mystery
# Sci-Fi
# IMAX
# Documentary War
# Musical
# Western
# Film-Noir


# These text are created for easiness purpose only since being checked multiple times
text.Adventure <- "Adventure"
text.Animation <- "Animation"
text.Children <- "Children"
text.Comedy <- "Comedy"
text.Fantasy <- "Fantasy"
text.Romance <- "Romance"
text.Drama <- "Drama"
text.Action <- "Action"
text.Crime <- "Crime"
text.Thriller <- "Thriller"
text.Horror <- "Horror"
text.Mystery <- "Mystery"
text.SciFi <- "Sci-Fi"
text.IMAX <- "IMAX"
text.Documentary <- "Documentary"
text.War <- "War"
text.Musical <- "Musical"
text.Western <- "Western"
text.FilmNoir <- "Film-Noir"

# Also removing movies whose score are missing in rating found not all movies have rating
# One way to handle NA value is to remove them using na.omit() function
# There is so much redundant code, try creating function and pass appropiate paramater
# Create dataframe sorted by score for each genre
movie.Adventure <- movie.group.table[movie.group.table$genres == text.Adventure,]
movie.Adventure <- na.omit(movie.Adventure)
movie.Adventure <- movie.Adventure[order(movie.Adventure$avg_score, decreasing = TRUE),]
```

```
movie.Adventure <- as.data.frame.matrix(movie.Adventure)

# Create dataframe sorted by score for each genre
movie.Animation <- movie.group.table[movie.group.table$genres == text.Animation,]
movie.Animation <- na.omit(movie.Animation)
movie.Animation <- movie.Animation[order(movie.Animation$avg_score, decreasing = TRUE),]
movie.Animation <- as.data.frame.matrix(movie.Animation)

# Create dataframe sorted by score for each genre
movie.Children <- movie.group.table[movie.group.table$genres == text.Children,]
movie.Children <- na.omit(movie.Children)
movie.Children <- movie.Children[order(movie.Children$avg_score, decreasing = TRUE),]
movie.Children <- as.data.frame.matrix(movie.Children)

# Create dataframe sorted by score for each genre
movie.Comedy <- movie.group.table[movie.group.table$genres == text.Comedy,]
movie.Comedy <- na.omit(movie.Comedy)
movie.Comedy <- movie.Comedy[order(movie.Comedy$avg_score, decreasing = TRUE),]
movie.Comedy <- as.data.frame.matrix(movie.Comedy)

# Create dataframe sorted by score for each genre
movie.Fantasy <- movie.group.table[movie.group.table$genres == text.Fantasy,]
movie.Fantasy <- na.omit(movie.Fantasy)
movie.Fantasy <- movie.Fantasy[order(movie.Fantasy$avg_score, decreasing = TRUE),]
movie.Fantasy <- as.data.frame.matrix(movie.Fantasy)

# Create dataframe sorted by score for each genre
movie.Romance <- movie.group.table[movie.group.table$genres == text.Romance,]
movie.Romance <- na.omit(movie.Romance)
movie.Romance <- movie.Romance[order(movie.Romance$avg_score, decreasing = TRUE),]
movie.Romance <- as.data.frame.matrix(movie.Romance)

# Create dataframe sorted by score for each genre
movie.Drama <- movie.group.table[movie.group.table$genres == text.Drama,]
movie.Drama <- na.omit(movie.Drama)
movie.Drama <- movie.Drama[order(movie.Drama$avg_score, decreasing = TRUE),]
movie.Drama <- as.data.frame.matrix(movie.Drama)

# Create dataframe sorted by score for each genre
movie.Action <- movie.group.table[movie.group.table$genres == text.Action,]
movie.Action <- na.omit(movie.Action)
movie.Action <- movie.Action[order(movie.Action$avg_score, decreasing = TRUE),]
movie.Action <- as.data.frame.matrix(movie.Action)

# Create dataframe sorted by score for each genre
movie.Crime <- movie.group.table[movie.group.table$genres == text.Crime,]
movie.Crime <- na.omit(movie.Crime)
movie.Crime <- movie.Crime[order(movie.Crime$avg_score, decreasing = TRUE),]
movie.Crime <- as.data.frame.matrix(movie.Crime)

# Create dataframe sorted by score for each genre
movie.Thriller <- movie.group.table[movie.group.table$genres == text.Thriller,]
movie.Thriller <- na.omit(movie.Thriller)
movie.Thriller <- movie.Thriller[order(movie.Thriller$avg_score, decreasing = TRUE),]
```

```
movie.Thriller <- as.data.frame.matrix(movie.Thriller)

# Create dataframe sorted by score for each genre
movie.Horror <- movie.group.table[movie.group.table$genres == text.Horror,]
movie.Horror <- na.omit(movie.Horror)
movie.Horror <- movie.Horror[order(movie.Horror$avg_score, decreasing = TRUE),]
movie.Horror <- as.data.frame.matrix(movie.Horror)

# Create dataframe sorted by score for each genre
movie.Mystery <- movie.group.table[movie.group.table$genres == text.Mystery,]
movie.Mystery <- na.omit(movie.Mystery)
movie.Mystery <- movie.Mystery[order(movie.Mystery$avg_score, decreasing = TRUE),]
movie.Mystery <- as.data.frame.matrix(movie.Mystery)

# Create dataframe sorted by score for each genre
movie.SciFi <- movie.group.table[movie.group.table$genres == text.SciFi,]
movie.SciFi <- na.omit(movie.SciFi)
movie.SciFi <- movie.SciFi[order(movie.SciFi$avg_score, decreasing = TRUE),]
movie.SciFi <- as.data.frame.matrix(movie.SciFi)

# Create dataframe sorted by score for each genre
movie.IMAX <- movie.group.table[movie.group.table$genres == text.IMAX,]
movie.IMAX <- na.omit(movie.IMAX)
movie.IMAX <- movie.IMAX[order(movie.IMAX$avg_score, decreasing = TRUE),]
movie.IMAX <- as.data.frame.matrix(movie.IMAX)

# Create dataframe sorted by score for each genre
movie.Documentary <- movie.group.table[movie.group.table$genres == text.Documentary,]
movie.Documentary <- na.omit(movie.Documentary)
movie.Documentary <- movie.Documentary[order(movie.Documentary$avg_score, decreasing =
    TRUE),]
movie.Documentary <- as.data.frame.matrix(movie.Documentary)

# Create dataframe sorted by score for each genre
movie.War <- movie.group.table[movie.group.table$genres == text.War,]
movie.War <- na.omit(movie.War)
movie.War <- movie.War[order(movie.War$avg_score, decreasing = TRUE),]
movie.War <- as.data.frame.matrix(movie.War)

# Create dataframe sorted by score for each genre
movie.Musical <- movie.group.table[movie.group.table$genres == text.Musical,]
movie.Musical <- na.omit(movie.Musical)
movie.Musical <- movie.Musical[order(movie.Musical$avg_score, decreasing = TRUE),]
movie.Musical <- as.data.frame.matrix(movie.Musical)


# Create dataframe sorted by score for each genre
movie.Western <- movie.group.table[movie.group.table$genres == text.Western,]
movie.Western <- na.omit(movie.Western)
movie.Western <- movie.Western[order(movie.Western$avg_score, decreasing = TRUE),]
movie.Western <- as.data.frame.matrix(movie.Western)

# Create dataframe sorted by score for each genre
movie.FilmNoir <- movie.group.table[movie.group.table$genres == text.FilmNoir,]
```

```
movie.FilmNoir <- na.omit(movie.FilmNoir)
movie.FilmNoir <- movie.FilmNoir[order(movie.FilmNoir$avg_score, decreasing = TRUE),]
movie.FilmNoir <- as.data.frame.matrix(movie.FilmNoir)


# ------------------ End of PART 2
    ----------------------------------------------------------------------

# ****************** Simple Testing, Let's see what we can recommend ********************

# Given userId, genre user likes, movie list user has watched find top-k movies from the
    genre
# which user has not watched and show titles and average score for the user to recommend

test_user <- classification.table1[10,c("userId", "genres", "movie_list")]
user.genre.label <- test_user$genres
k <- 4

# Using if else loop depending on genre user has been labelled we need to check
    respective dataframe

recommendations <- data.frame(Title=character(),MovieId=numeric(),Score=double())
if (user.genre.label == text.Adventure)
{
  for (id in movie.Adventure$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Adventure [movie.Adventure$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Animation) {
  for (id in movie.Animation$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Animation[movie.Animation$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }

} else if (user.genre.label == text.Children) {

  for (id in movie.Children$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
```

```
    record <- movie.Children[movie.Adventure$movieId == id, ]
    newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
        avg_score)
    recommendations <- rbind(recommendations, newRow)
  }
  if (nrow(recommendations) >= k) break
}

} else if (user.genre.label == text.Comedy) {
  for (id in movie.Comedy$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Comedy[movie.Comedy$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break

  }
} else if (user.genre.label == text.Fantasy) {
  for (id in movie.Fantasy$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Fantasy[movie.Fantasy$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break

  }
} else if (user.genre.label == text.Romance) {
  for (id in movie.Romance$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Romance[movie.Romance$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break

  }

} else if (user.genre.label == text.Drama) {
  for (id in movie.Drama$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Drama[movie.Drama$movieId == id, ]
```

```
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break

  }
} else if (user.genre.label == text.Action) {
  for (id in movie.Action$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Action[movie.Action$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break

  }

} else if (user.genre.label == text.Crime) {
  for (id in movie.Crime$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Crime[movie.Crime$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Thriller) {
  # Code for Thriller genre
  for (id in movie.Thriller$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Thriller[movie.Thriller$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score = record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }

} else if (user.genre.label == text.Horror) {
  for (id in movie.Horror$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Horror[movie.Horror$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
```

```
        avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Mystery) {
  for (id in movie.Mystery$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Mystery[movie.Mystery$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.SciFi) {
  for (id in movie.SciFi$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.SciFi[movie.SciFi$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.IMAX) {
  for (id in movie.IMAX$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.IMAX[movie.IMAX$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Documentary) {
  for (id in movie.Documentary$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Documentary[movie.Documentary$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.War) {
```

```
  for (id in movie.War$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.War[movie.War$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Musical){
  for (id in movie.Musical$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Musical[movie.Musical$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
} else if (user.genre.label == text.Western){
  for (id in movie.Western$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.Western[movie.Western$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
}else if (user.genre.label == text.FilmNoir){
  for (id in movie.FilmNoir$movieId)
  {
    if (!(id %in% test_user$movie_list))
    {
      record <- movie.FilmNoir[movie.FilmNoir$movieId == id, ]
      newRow <- data.frame(Title = record$title, MovieId = record$movieId, Score =record$
          avg_score)
      recommendations <- rbind(recommendations, newRow)
    }
    if (nrow(recommendations) >= k) break
  }
}else{
  print("Unknown Genre or no Genre")
}

#to show the recommendations
  observeEvent(input$testing, {
```

```
    output$table <- renderDataTable(
      recommendations)
  })
})
```