MIT Academy of Engineering

(An Autonomous Institute affiliated to Savitribai Phule Pune University)

SEMINAR REPORT

On

Movie Recommendation System

By

Sachin Kumar

Shubham Agrahari

Guided by

Mrs. Sunita Barve

Mrs. Sharmila Kharat

Mrs. Diptee Chikmurge

DEPARTMENT OF COMPUTER ENGINEERING

MIT ACADEMY OF ENGINEERING

ALANDI (D), PUNE

2016 - 2017

MIT ACADEMY OF ENGINEERING
ALANDI (D), PUNE
DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the seminar entitled "**Movie Recommendation System**" has been carried out by "**Sachin Kumar & Shubham Agrahari**" under my guidance in partial fulfillment of Third Year Computer Engineering of Savitribai Phule Pune University, Pune during the academic year 2016-2017.

Mrs. Sunita Barve
Seminar Guide

Dr. S.A. Jain
Head of the Department

ACKNOWLEDGEMENT

I would like to take this opportunity to express my profound gratitude to all those who have provided timely guidance and helping hand in making my Project a success.

I want to thank the department of Computer Engineering of MITAOE for giving me opportunity to represent this Project as part of third year to do the necessary work and to use departmental resources. I extend sincere thanks to respected **Mr. Shital Kumar Jain Sir (HOD)** of Computer Department & all the teachers of the department who encouraged me for this Project.

I deeply thanks to my guide **Mrs. Sunita Barve** mam whose guidance, stimulating suggestions and encouragement helped me in all the time of discussion from the inception of the project till its completion . From this Project my guide has brought the best out of us.

I even want to thank our department colleagues for all their helps, support, interest and valuable hints.

(Shubham Agrahari & Sachin Kumar)

INDEX

FIGURE INDEX

ABSTRACT

Recommendation systems make suggestions about artifacts to a user. For instance, they may predict whether a user would be interested in seeing a particular movie. Recommendation methods collect ratings of artifacts from many individuals, and use nearest-neighbor techniques to make recommendations to a user concerning new artifacts. However, these methods do not use the significant amount of other information that is often available about the nature of each artifact -- such as cast lists or movie reviews, for example. This paper presents an inductive learning approach to recommendation that is able to use both ratings information and other forms of information about each artifact in predicting user preferences.

We will build a Building a Movie Recommendation System based on selected training sets provided by Movie Lens. Several machine learning related algorithms – collaborative filtering recommender (CFR), KNN. Predict the rating from particular users for unrated movies. RMSE (Root-Mean-Square-Error) is applied as the main criteria to evaluate their performance. The simulation result shows distinct performance due to selected algorithms as well as the corresponding learning rates.

Keywords*:* Movie recommendation, Collaborative filtering, Genetic algorithms, K-means.

CHAPTER -1

INTRODUCTION

1.1 Problem Statement

Building a Movie Recommendation System. I develop a collaborative filtering recommender (CFR) system for recommending movies. The basic idea of CFR systems is that, if two users share the same interests in the past, e.g. they liked the same book or the same movie, they will also have similar tastes in the future. If, for example, user A and user B have a similar purchase history and user A recently bought a book that user B has not yet seen, the basic idea is to propose this book to user B.

1.2 Related Work

Movie Lens is a data set that provides 10000054 user ratings on movies.95580 tags applied to 10681 movies by 71567 users. Users of Movie Lens were selected randomly. All users rated at least 50 movies. Each user represented by a unique id.
And Each movie is represented by unique id.1st we start our work with data analysis of data set provided by movie lens, convert raw data into useful data by following:---

1.2.1 Data Pre-processing

Some pre-processing of the data available is required before creating the recommendation system.

First of all, I will re-organize the information of movie genres in such a way that allows future users to search for the movies they like within specific genres. From the design perspective, this is much easier for the user compared to selecting a movie from a single very long list of all the available movies.

1.2.2 Exploring Similarity Data

Collaborative filtering algorithms are based on measuring the similarity between users or between items. For this purpose, *recommenderlab* contains the similarity function. The supported methods to compute similarities are *cosine*, *pearson*, and *jaccard*.

1.2.3 Data Preparation

The data preparation process consists of the following steps:

1. Select the relevant data.
2. Normalize the data.
3. Binarize the data.

1. Select The Relevant Data

In order to select the most relevant data, I define the minimum number of users per rated movie as 50 and the minimum views number per movie as 50.

2. Normalize The Data

Having users who give high (or low) ratings to all their movies might bias the results. In order to remove this effect, I normalize the data in such a way that the average rating of each user is 0. As a quick check, I calculate the average rating by users, and it is equal to 0.

3. Binarizing Data

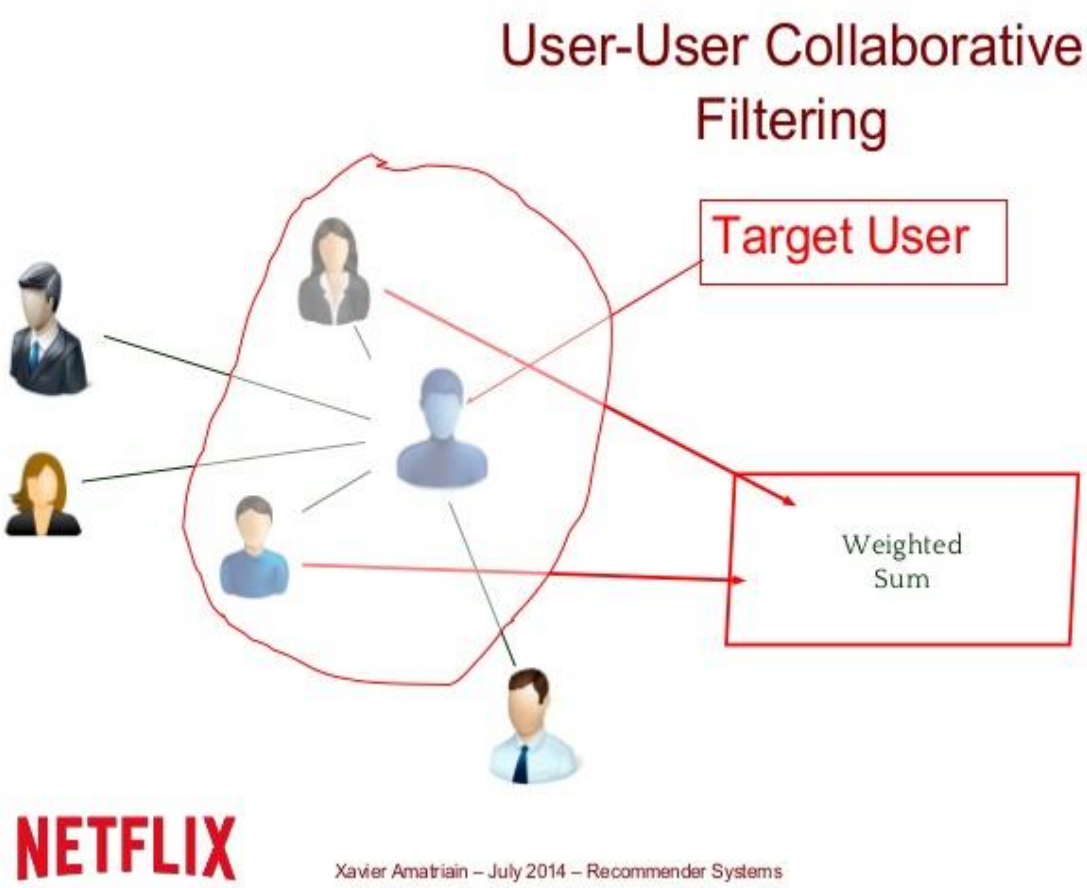
Some recommendation models work on binary data, so it might be useful to binarize the data, that is, define a table containing only 0s and 1s. The 0s will be either treated as missing values or as bad ratings.
In our case, I can either:

- Define a matrix having 1 if the user rated the movie, and 0 otherwise. In this case, the information about the rating is lost.
- Define a matrix having 1 if the rating is above or equal to a definite threshold (for example, 3), and 0 otherwise. In this case, giving a bad rating to a movie is equivalent to not having rated it.

1.2.4 Movie Recommendation Systems based on Collaborative Filtering

Recommendation systems (RS), introduced by Tapestry project in 1992, is one of the most successful information management systems . The practical recommender applications help users to filter mass useless information for dealing with the information overloading and providing personalized suggestions. There has been a great success in e-commerce to make the customer access the preferred products, and improve the business profit. In addition, to enhance the ability of personalization, recommendation system is also widely deployed in many multimedia websites for targeting media products to particular

customers. Nowadays, Collaborative filtering (CF) is the most effective technique employed



by movie recommendation systems, which is on the basis of the nearest-

Fig 1.1 user base collaborative filtering

neighbor mechanism. It is on the assumption that people who have similar history rating pattern may be on the maximum likelihood that have the same preference in the future. All "likeminded" users, called neighbors, are derived from their rating database that is recording evaluation values to movies. The prediction of a missing rating given by a target user can be inferred by the weighted similarity of his/her neighborhood. Divides CF techniques into two important classes of recommender systems: memorybased CF and model-based CF. Memory-based CF operate on the entire user space to search nearest neighbors for an active user, and automatically produce a list of suggested movies to recommend. This method suffers from the computation complexity and data sparsity problem. In order to address CF in which the correlations between items are computed to form the neighborhood for a target

computational and memory bottleneck issues, Sarwar et al. proposed an item-based item [4]. In their empirical studies, it is proved that item-based approach can shorten computation time apparently while providing comparable prediction accuracy. Model-based CF, on the other hand, develops a prebuild model to store rating patterns based on user-rating database which can deal with the scalability and sparsity issues. In terms of recommendation quality, model-based CF applications can perform as well as memory-based ones. However, model-based approaches are time consuming in building and training the offline model which is hard to be updated as well. Algorithms that often used in model-based CF applications include Bayesian networks , clustering algorithms , neural networks , and SVD (Singular Value Decomposition) . While traditional collaborative recommendation systems have their instinct limitations, such as computational scalability, data sparsity and cold start, and these issues are still challenges that affect the prediction quality. Over the last decade, there have been high interests toward RS area due to the possible improvement in performance and problems solving capability.

1.2.5 K-means clustering

K-means algorithm is one of the most commonly used clustering approaches due to its simplicity, flexibility and computation efficiency especially considering large amounts of data. K-means iteratively computes k cluster centers to assign objects into the most nearest cluster based on distance measure. When center points have no more change, the clustering algorithm comes to a convergence. However, K-means lacks the ability of selecting appropriate initial seed and may lead to inaccuracy of classification. Randomly selecting initial seed could result in a local optimal solution that is quite inferior to the global optimum. In other words, different initial seeds running on the same dataset may produce different partition results.

Given a set of objects ($x_1, x_2, \ldots\ldots x_n$), where each object is an m-dimensional vector, K-means algorithm aims to automatically partition these objects into k groups. Typically, the procedure consists of the following steps [24-25]:

1) choose k initial cluster centers Cj, j=1,2,3…k;
2) each xi is assigned to its closest cluster center according to the distance metric;
3) compute the sum of squared distances from all members in one cluster:

$$\boldsymbol{J = \Sigma_{j=1}^{k} \Sigma_{i \in C_{temp}} \left|\left| x_i - M_j \right|\right|^2}$$

where Mj denotes the mean of data points in C_{temp};

4) if there is no further change, then the algorithm has converged and clustering task is end; otherwise, recalculate the Mj of k clusters as the new cluster centers and go to step2. To overcome the above limitations, we introduce genetic algorithm to merge with K-means clustering process for the enhancement of classification quality around a specified k.

1.3 Project Purpose and Goal

This thesis project consists of three main parts. One is a general study of and theory behind social networks and recommender systems, the second is the development of a prototype application on a social networking site and the third is the evaluation of a recommender algorithm that has been developed at Ericsson Research. Three research questions were chosen for the project:

1.How well does the recommender system predict ratings?

2.Can the existing algorithm be revised in order to optimize the recommendations?

3.Is a recommender system based on social networks value able?

The literature study was done to provide insight in to the history and world of recommender systems and social networks. As a step in evaluating the algorithm, Ericsson wished to have an application for movie recommendations built on a social networking site, where users could interact with the recommender system. The purpose of the prototyping activities was to gather the data necessary to evaluate the accuracy of the algorithm as well as gather insight into the users' perspectives on the recommender system. One hypothesis in this project has been that the algorithm returns better recommendations than if random movies were returned. The goal of the project has been to suggest changes which can further the development of the recommender system

1.4 Motivation

In this project, I develop a collaborative filtering recommender (CFR) system for recommending movies.

The basic idea of CFR systems is that, if two users share the same interests in the past, e.g. they liked the same book or the same movie, they will also have similar tastes in the future. If, for example, user A and user B have a similar purchase history and user A recently bought a book that user B has not yet seen, the basic idea is to propose this book to user B.

The collaborative filtering approach considers only user preferences and does not take into account the features or contents of the items (books or movies) being recommended. In this project, in order to recommend movies I will use a large set of users preferences towards the movies from a publicly available movie rating dataset.

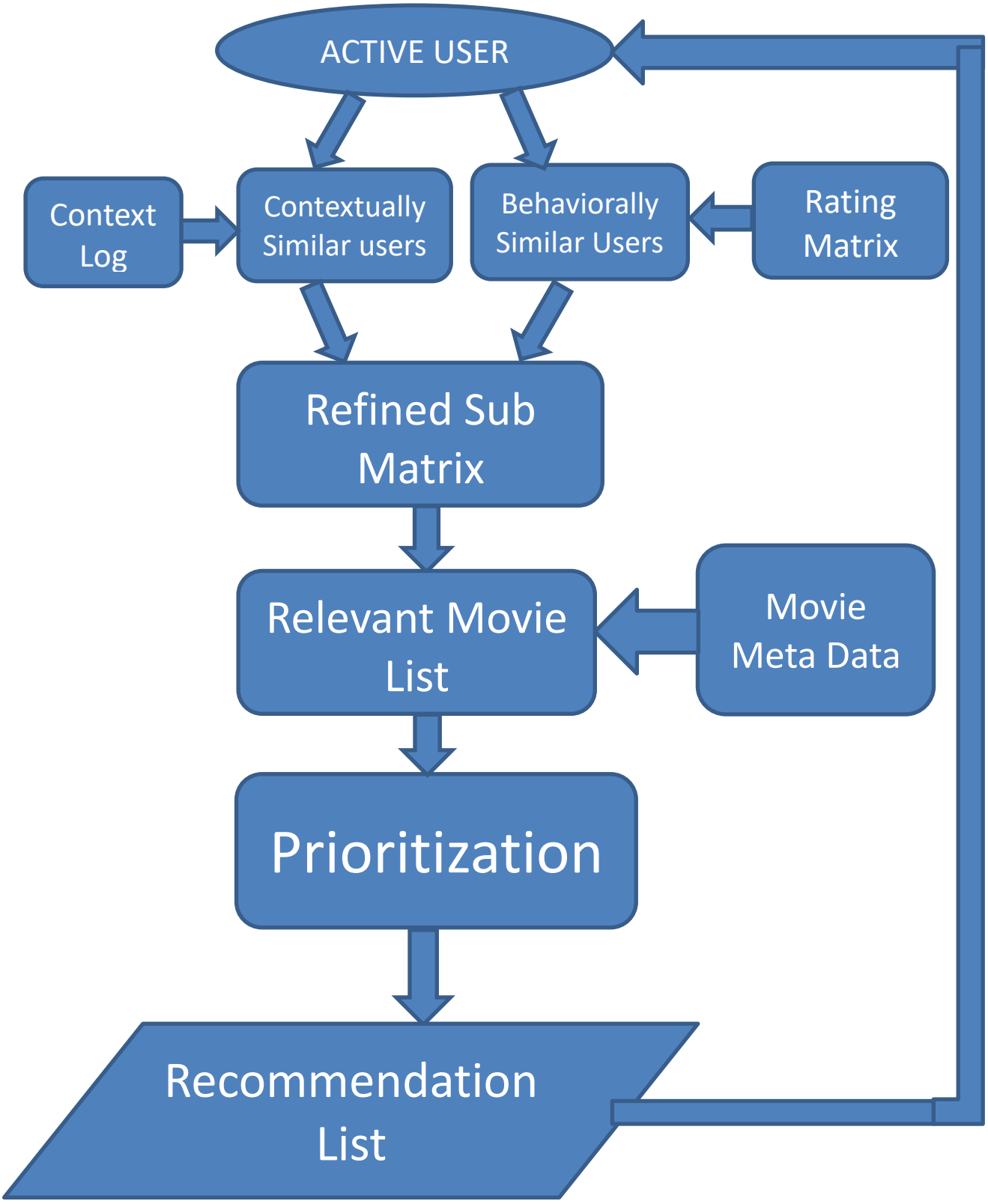
The dataset used was from Movie Lens, and is publicly available at http://grouplens.org/datasets/movielens/latest. In order to keep the recommender simple, I used the smallest dataset available (ml-latest-small.zip), which at the time of download contaied 105339 ratings and 6138 tag applications across 103 movies.

These data were created by 668 users between April 03, 1996 and January 09, 2016. This dataset was generated on January 11, 2016.

CHAPTER -2

DESIGN DETAILS

2.1 DFD(Data Flow Diagram)



2.2 Machine Learning parallel Training Block Diagram

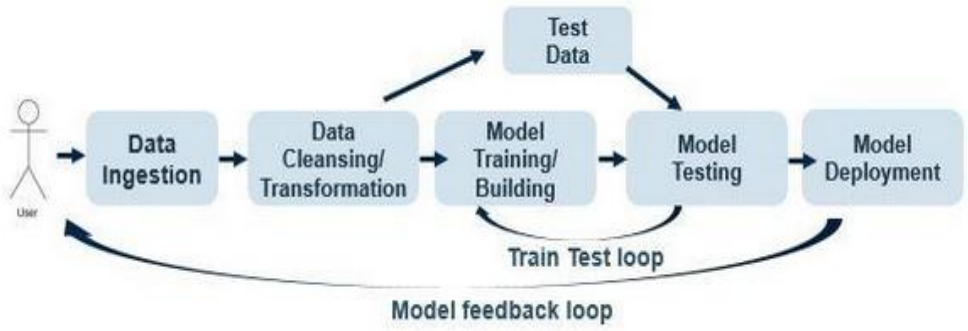


Fig. 2.2 Machine Learning parallel Training Block Diagram

Multi-task learning (MTL) is a subfield of machine learning in which multiple learning tasks are solved at the same time, while exploiting commonalities and differences across tasks. This can result in improved learning efficiency and prediction accuracy for the task-specific models, when compared to training the models separately. Multitask Learning is an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias. It does this by learning tasks in parallel while using a shared representation; what is learned for each task can help other tasks be learned better.

CHAPTER -3

3.1 Hardware & Software requirements

✓ Hardware Requirements :

- i5 processor
- 4gb RAM
- 100gb memory

✓ Software requirements :

- r studio
- Operating System: windows 7,linux,windows 8

3.2 Environmental Setup Steps

The open-source Rstudio environment makes it easy to write code and upload it to the i/o board. It runs on Windows, Mac OS X, and Linux.

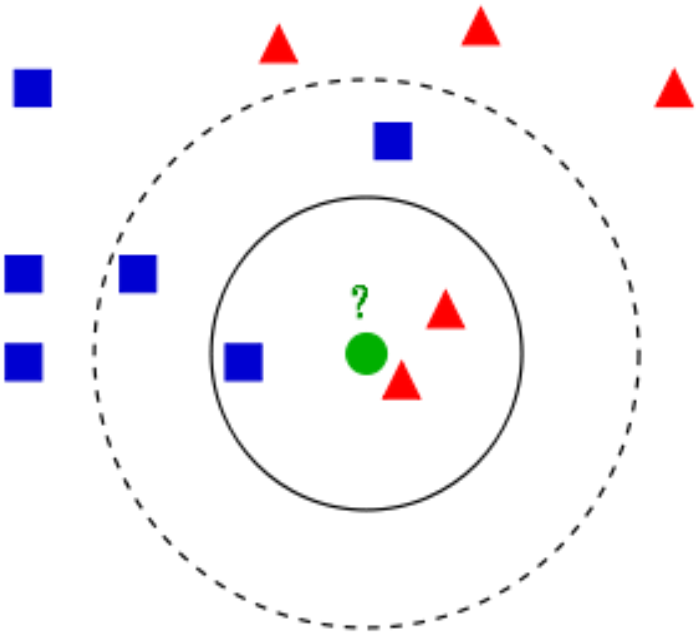
3.3 Algorithms

3.3.1 KNN Algorithm

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, *k* is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the **overlap metric** (or Hamming distance). In the context of gene expression data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman.[1] Often, the classification accuracy of *k*-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.



A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbors due to their large number.[4] One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its *k* nearest neighbors. The class (or value, in regression problems) of each of the *k* nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. *K*-NN can then be applied to the SOM.

3.4 Code

_____helpercode.R_____

__

```
library(proxy)
library(recommenderlab)
library(reshape2)
movies <- read.csv("movies.csv", header = TRUE, stringsAsFactors=FALSE)
ratings <- read.csv("ratings.csv", header = TRUE)
movies2 <- movies[-which((movies$movieId %in% ratings$movieId) ==
FALSE),]
```

```
movie_recommendation <- function(input,input2,input3) {
  #input = "Gladiator (2000)"
  #input2 = "Aeon Flux (2005)"
  #input3 = "Alexander (2004)"
  row_num <- which(movies2[,2] == input)
  row_num2 <- which(movies2[,2] == input2)
  row_num3 <- which(movies2[,2] == input3)
  userSelect <- matrix(NA,10325)
  userSelect[row_num] <- 5 #hard code first selection to rating 5
  userSelect[row_num2] <- 4 #hard code second selection to rating 4
  userSelect[row_num3] <- 3 #hard code third selection to rating 3
  userSelect <- t(userSelect)

  ratingmat <- dcast(ratings, userId~movieId, value.var = "rating", na.rm=FALSE)
  ratingmat <- ratingmat[,-1]
  colnames(userSelect) <- colnames(ratingmat)
  ratingmat2 <- rbind(userSelect,ratingmat)
  ratingmat2 <- as.matrix(ratingmat2)

  #Convert rating matrix into a sparse matrix
  ratingmat2 <- as(ratingmat2, "realRatingMatrix")

  #Create Recommender Model. "UBCF" stands for user-based collaborative
filtering
  recommender_model <- Recommender(ratingmat2, method =
"UBCF",param=list(method="Cosine",nn=30))
  recom <- predict(recommender_model, ratingmat2[1], n=10)
  recom_list <- as(recom, "list")
  no_result <- data.frame(matrix(NA,1))
  recom_result <- data.frame(matrix(NA,10))
  if (as.character(recom_list[1])=='character(0)'){
    no_result[1,1] <- "Sorry, there is not enough information in our database on the
movies you've selected. Try to select different movies you like."
    colnames(no_result) <- "No results"
    return(no_result)
  } else {
    for (i in c(1:10)){
      recom_result[i,1] <- as.character(subset(movies,
                                               movies$movieId ==
as.integer(recom_list[[1]][i]))$title)
    }
```

```
colnames(recom_result) <- "User-Based Collaborative Filtering Recommended
Titles"
 return(recom_result)
 }
}
```

_____ui.R_____

```
library(shiny)

genre_list <- c("Action", "Adventure", "Animation", "Children",
               "Comedy", "Crime","Documentary", "Drama", "Fantasy",
               "Film.Noir", "Horror", "Musical", "Mystery","Romance",
               "Sci.Fi", "Thriller", "War", "Western")

shinyUI(fluidPage(
 titlePanel("Movie Recommendation System"),
 fluidRow(

  column(4, h3("Select Movie Genres You Prefer (order matters):"),
       wellPanel(
        selectInput("input_genre", "Genre #1",
                 genre_list),
        selectInput("input_genre2", "Genre #2",
                 genre_list),
        selectInput("input_genre3", "Genre #3",
                 genre_list)
        #submitButton("Update List of Movies")
```

```
        )),
column(4, h3("Select Movies You Like of these Genres:"),
          wellPanel(
            # This outputs the dynamic UI component
            uiOutput("ui"),
            uiOutput("ui2"),
            uiOutput("ui3")
            #submitButton("Get Recommendations")
          )),

    column(4,
          h3("You Might Like The Following Movies Too!"),
          tableOutput("table")
          #verbatimTextOutput("dynamic_value")
    )
  ),

  fluidRow(
    column(12,
          helpText("For a detailed description of this project please visit",
                a("the link", href="http://rpubs.com/jeknov/movieRec",
target="_blank")),
          helpText("For a code, press", a("here", href =
"https://github.com/jeknov/movieRec", target="_blank"))
            ) )
))
```

_____server.R_____

```
library(shiny)
library(proxy)
library(recommenderlab)
library(reshape2)
source("helpercode.R")

search <- read.csv("search.csv", stringsAsFactors=FALSE)
ratings <- read.csv("ratings.csv", header = TRUE)
search <- search[-which((search$movieId %in% ratings$movieId) == FALSE),]

formatInput <- function(v,a,d){
 ## This function formats the user's input of Valence-Arousal-Dominance
 ## and outputs them as a vector

 c(v,a,d)
}

shinyServer(function(input, output) {

 output$ui <- renderUI({
  if (is.null(input$input_genre))
   return()
```

```
 switch(input$input_genre,
        "Action" = selectInput("select", "Movie of Genre #1",
                               choices = sort(subset(search, Action == 1)$title),
                               selected = sort(subset(search, Action == 1)$title)[1]),
        "Adventure" = selectInput("select", "Movie of Genre #1",
                                  choices = sort(subset(search, Adventure == 1)$title),
                                  selected = sort(subset(search, Adventure == 1)$title)[1]),
        "Animation" =  selectInput("select", "Movie of Genre #1",
                                   choices = sort(subset(search, Animation == 1)$title),
                                   selected = sort(subset(search, Animation == 1)$title)[1]),
        "Children" =  selectInput("select", "Movie of Genre #1",
                                  choices = sort(subset(search, Children == 1)$title),
                                  selected = sort(subset(search, Children == 1)$title)[1]),
        "Comedy" =  selectInput("select", "Movie of Genre #1",
                                choices = sort(subset(search, Comedy == 1)$title),
                                selected = sort(subset(search, Comedy == 1)$title)[1]),
        "Crime" =  selectInput("select", "Movie of Genre #1",
                               choices = sort(subset(search, Crime == 1)$title),
                               selected = sort(subset(search, Crime == 1)$title)[1]),
        "Documentary" =  selectInput("select", "Movie of Genre #1",
                                     choices = sort(subset(search, Documentary == 1)$title),
                                     selected = sort(subset(search, Documentary ==
1)$title)[1]),
        "Drama" =  selectInput("select", "Movie of Genre #1",
                               choices = sort(subset(search, Drama == 1)$title),
                               selected = sort(subset(search, Drama == 1)$title)[1]),
```

```
"Fantasy" =  selectInput("select", "Movie of Genre #1",
                         choices = sort(subset(search, Fantasy == 1)$title),
                         selected = sort(subset(search, Fantasy == 1)$title)[1]),
 "Film.Noir" =  selectInput("select", "Movie of Genre #1",
                           choices = sort(subset(search, Film.Noir == 1)$title),
                           selected = sort(subset(search, Film.Noir == 1)$title)[1]),
 "Horror" =  selectInput("select", "Movie of Genre #1",
                         choices = sort(subset(search, Horror == 1)$title),
                         selected = sort(subset(search, Horror == 1)$title)[1]),
 "Musical" =  selectInput("select", "Movie of Genre #1",
                          choices = sort(subset(search, Musical == 1)$title),
                          selected = sort(subset(search, Musical == 1)$title)[1]),
 "Mystery" =  selectInput("select", "Movie of Genre #1",
                          choices = sort(subset(search, Mystery == 1)$title),
                          selected = sort(subset(search, Mystery == 1)$title)[1]),
 "Romance" =  selectInput("select", "Movie of Genre #1",
                          choices = sort(subset(search, Romance == 1)$title),
                          selected = sort(subset(search, Romance == 1)$title)[1]),
 "Sci.Fi" =  selectInput("select", "Movie of Genre #1",
                         choices = sort(subset(search, Sci.Fi == 1)$title),
                         selected = sort(subset(search, Sci.Fi == 1)$title)[1]),
 "Thriller" =  selectInput("select", "Movie of Genre #1",
                           choices = sort(subset(search, Thriller == 1)$title),
                           selected = sort(subset(search, Thriller == 1)$title)[1]),
 "War" =  selectInput("select", "Movie of Genre #1",
                      choices = sort(subset(search, War == 1)$title),
```

```
        "Western" = selectInput("select", "Movie of Genre #1",
         selected = sort(subset(search, War == 1)$title)[1]),
 choices = sort(subset(search, Western == 1)$title),
                          selected = sort(subset(search, Western == 1)$title)[1])
  )
})

output$ui2 <- renderUI({
  if (is.null(input$input_genre2))
    return()

  switch(input$input_genre2,
         "Action" = selectInput("select2", "Movie of Genre #2",
                                choices = sort(subset(search, Action == 1)$title),
                                selected = sort(subset(search, Action == 1)$title)[1]),
         "Adventure" = selectInput("select2", "Movie of Genre #2",
                                   choices = sort(subset(search, Adventure == 1)$title),
                                   selected = sort(subset(search, Adventure == 1)$title)[1]),
         "Animation" =  selectInput("select2", "Movie of Genre #2",
                                    choices = sort(subset(search, Animation == 1)$title),
                                    selected = sort(subset(search, Animation == 1)$title)[1]),
         "Children" =  selectInput("select2", "Movie of Genre #2",
                                   choices = sort(subset(search, Children == 1)$title),
                                   selected = sort(subset(search, Children == 1)$title)[1]),
         "Comedy" =  selectInput("select2", "Movie of Genre #2",
                                 choices = sort(subset(search, Comedy == 1)$title),
```

```
                selected = sort(subset(search, Comedy == 1)$title)[1]),
              "Crime" =  selectInput("select2", "Movie of Genre #2",
                                     choices = sort(subset(search, Crime == 1)$title),
                                     selected = sort(subset(search, Crime == 1)$title)[1]),
               "Documentary" =  selectInput("select2", "Movie of Genre #2",
                                        choices = sort(subset(search, Documentary == 1)$title),
                                        selected = sort(subset(search, Documentary ==
1)$title)[1]),
               "Drama" =  selectInput("select2", "Movie of Genre #2",
                                     choices = sort(subset(search, Drama == 1)$title),
                                     selected = sort(subset(search, Drama == 1)$title)[1]),
              "Fantasy" =  selectInput("select2", "Movie of Genre #2",
                                       choices = sort(subset(search, Fantasy == 1)$title),
                                       selected = sort(subset(search, Fantasy == 1)$title)[1]),
              "Film.Noir" =  selectInput("select2", "Movie of Genre #2",
                                         choices = sort(subset(search, Film.Noir == 1)$title),
                                         selected = sort(subset(search, Film.Noir == 1)$title)[1]),
              "Horror" =  selectInput("select2", "Movie of Genre #2",
                                      choices = sort(subset(search, Horror == 1)$title),
                                      selected = sort(subset(search, Horror == 1)$title)[1]),
              "Musical" =  selectInput("select2", "Movie of Genre #2",
                                       choices = sort(subset(search, Musical == 1)$title),
                                       selected = sort(subset(search, Musical == 1)$title)[1]),
              "Mystery" =  selectInput("select2", "Movie of Genre #2",
                                       choices = sort(subset(search, Mystery == 1)$title),
                                       selected = sort(subset(search, Mystery == 1)$title)[1]),
```

```
                    "Romance" =  selectInput("select2", "Movie of Genre #2",
                                 choices = sort(subset(search, Romance == 1)$title),
                                 selected = sort(subset(search, Romance == 1)$title)[1]),
        "Sci.Fi" =  selectInput("select2", "Movie of Genre #2",
                                choices = sort(subset(search, Sci.Fi == 1)$title),
                                selected = sort(subset(search, Sci.Fi == 1)$title)[1]),
        "Thriller" =  selectInput("select2", "Movie of Genre #2",
                                  choices = sort(subset(search, Thriller == 1)$title),
                                  selected = sort(subset(search, Thriller == 1)$title)[1]),
        "War" =  selectInput("select2", "Movie of Genre #2",
                             choices = sort(subset(search, War == 1)$title),
                             selected = sort(subset(search, War == 1)$title)[1]),
        "Western" = selectInput("select2", "Movie of Genre #2",
                                choices = sort(subset(search, Western == 1)$title),
                                selected = sort(subset(search, Western == 1)$title)[1])
  )
})

output$ui3 <- renderUI({
  if (is.null(input$input_genre3))
    return()

  switch(input$input_genre3,
        "Action" = selectInput("select3", "Movie of Genre #3",
                               choices = sort(subset(search, Action == 1)$title),
```

```
                selected = sort(subset(search, Action == 1)$title)[1]),
        "Adventure" = selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Adventure == 1)$title),
                selected = sort(subset(search, Adventure == 1)$title)[1]),
        "Animation" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Animation == 1)$title),
                selected = sort(subset(search, Animation == 1)$title)[1]),
        "Children" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Children == 1)$title),
                selected = sort(subset(search, Children == 1)$title)[1]),
        "Comedy" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Comedy == 1)$title),
                selected = sort(subset(search, Comedy == 1)$title)[1]),
        "Crime" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Crime == 1)$title),
                selected = sort(subset(search, Crime == 1)$title)[1]),
        "Documentary" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Documentary == 1)$title),
                selected = sort(subset(search, Documentary ==
1)$title)[1]),
        "Drama" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Drama == 1)$title),
                selected = sort(subset(search, Drama == 1)$title)[1]),
        "Fantasy" =  selectInput("select3", "Movie of Genre #3",
                choices = sort(subset(search, Fantasy == 1)$title),
                selected = sort(subset(search, Fantasy == 1)$title)[1]),
```

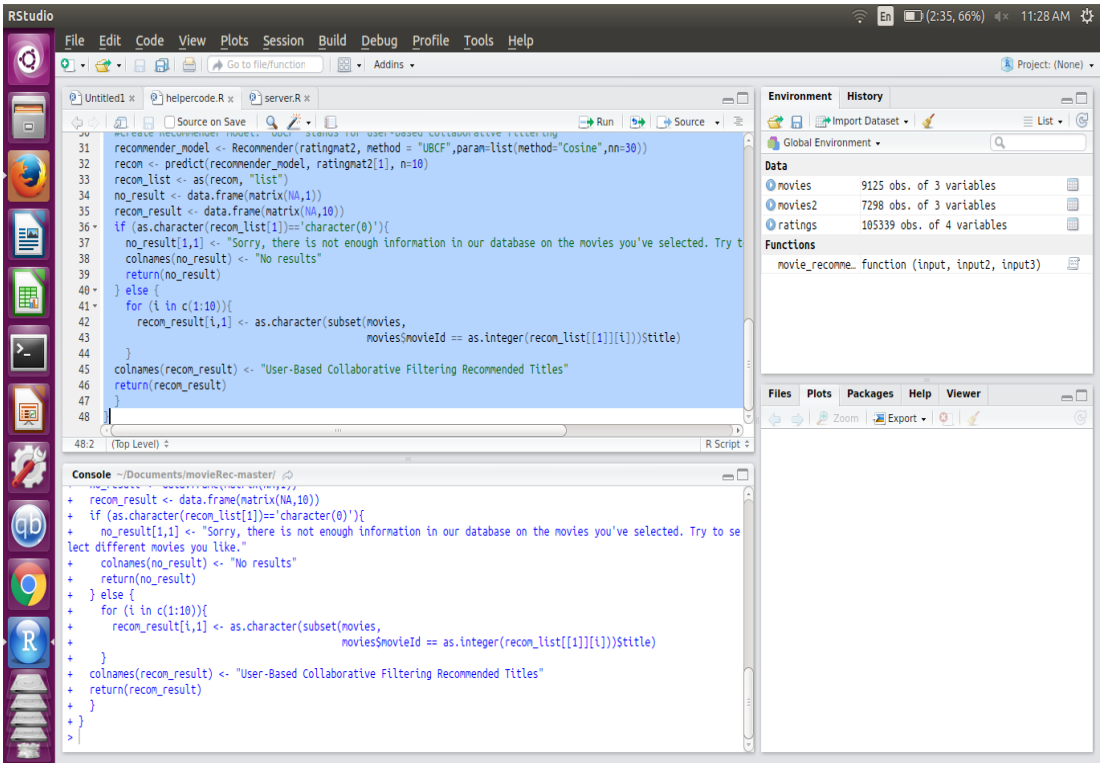
```
            "Film.Noir" =  selectInput("select3", "Movie of Genre #3",
                           choices = sort(subset(search, Film.Noir == 1)$title),
                            selected = sort(subset(search, Film.Noir == 1)$title)[1]),
"Horror" =  selectInput("select3", "Movie of Genre #3",
                          choices = sort(subset(search, Horror == 1)$title),
                          selected = sort(subset(search, Horror == 1)$title)[1]),
"Musical" =  selectInput("select3", "Movie of Genre #3",
                           choices = sort(subset(search, Musical == 1)$title),
                           selected = sort(subset(search, Musical == 1)$title)[1]),
"Mystery" =  selectInput("select3", "Movie of Genre #3",
                           choices = sort(subset(search, Mystery == 1)$title),
                           selected = sort(subset(search, Mystery == 1)$title)[1]),
"Romance" =  selectInput("select3", "Movie of Genre #3",
                           choices = sort(subset(search, Romance == 1)$title),
                           selected = sort(subset(search, Romance == 1)$title)[1]),
"Sci.Fi" =  selectInput("select3", "Movie of Genre #3",
                          choices = sort(subset(search, Sci.Fi == 1)$title),
                          selected = sort(subset(search, Sci.Fi == 1)$title)[1]),
"Thriller" =  selectInput("select3", "Movie of Genre #3",
                           choices = sort(subset(search, Thriller == 1)$title),
                           selected = sort(subset(search, Thriller == 1)$title)[1]),
"War" =  selectInput("select3", "Movie of Genre #3",
                         choices = sort(subset(search, War == 1)$title),
                         selected = sort(subset(search, War == 1)$title)[1]),
"Western" = selectInput("select3", "Movie of Genre #3",
```

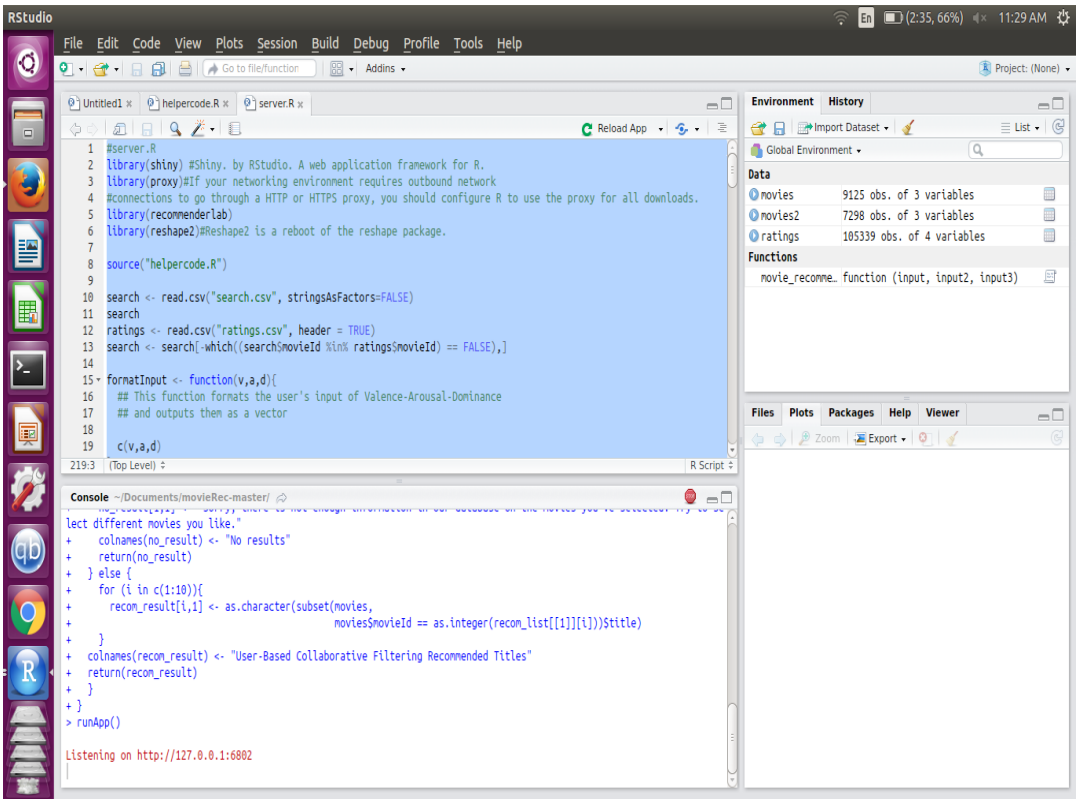
```
                choices = sort(subset(search, Western == 1)$title),
                selected = sort(subset(search, Western == 1)$title)[1])
  )
})

output$table <- renderTable({
  movie_recommendation(input$select, input$select2, input$select3)
})

Output :$dynamic_value <- renderPrint({
                                    c(input$select,input$select2,input$select3)
                                                    })
 })
```

3.4 Compilation Steps

Shiny applications present some challenges for the debugger because the breakpoints can't be set until the application is executed; the function objects that need to have breakpoints injected don't exist until then.

For this reason, breakpoints in Shiny applications *only* work inside the shiny Server function. Breakpoints are not currently supported in the user interface (i.e. ui.R), globals (i.e. global.R), or other .R sources used in Shiny applications. This may be improved in a future version of RStudio.

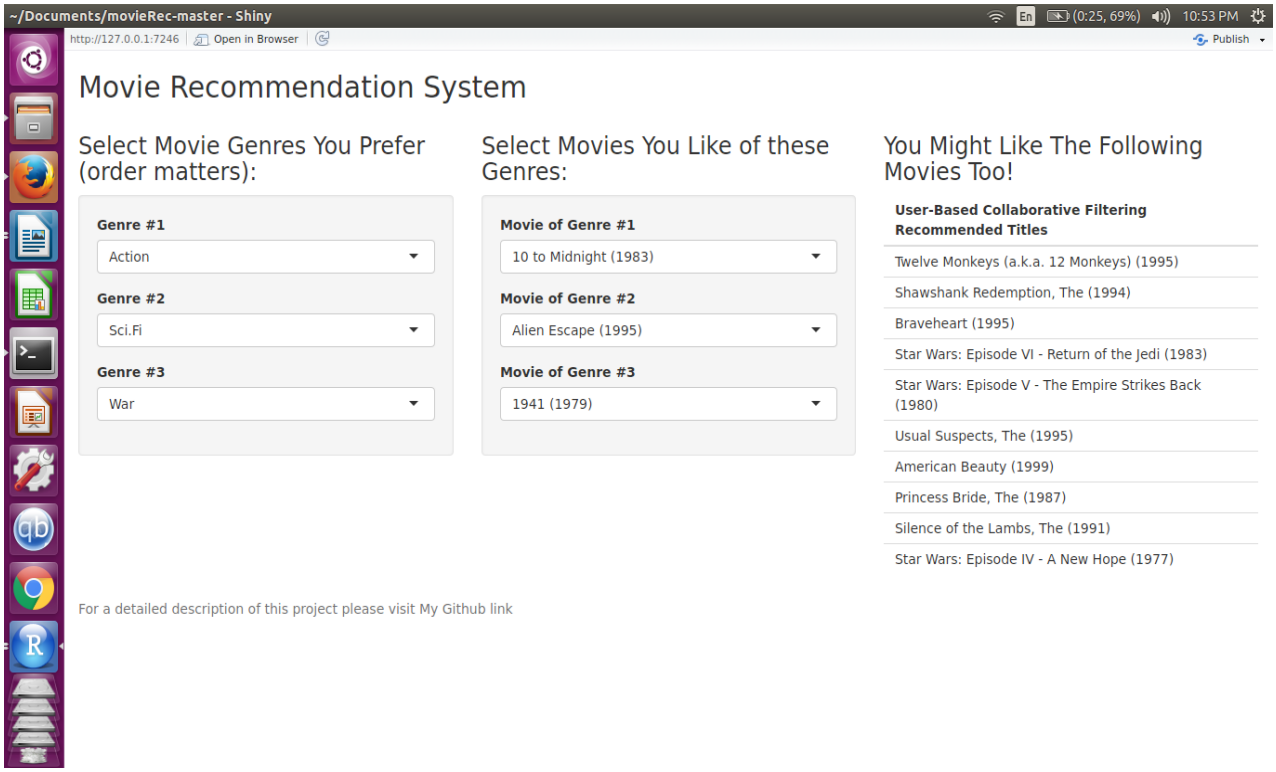

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1 helpercode.R server.R
Source on Save Run Source
recommender_model <- Recommender(ratingmat2, method = "UBCF",paran=list(method="Cosine",nn=30))
recom <- predict(recommender_model, ratingmat2[1], n=10)
recom_list <- as(recom, "list")
no_result <- data.frame(matrix(NA,1))
recom_result <- data.frame(matrix(NA,10))
if (as.character(recom_list[1])=='character(0)'){
no_result[1,1] <- "Sorry, there is not enough information in our database on the movies you've selected. Try t
colnames(no_result) <- "No results"
return(no_result)
} else {
for (i in c(1:10)){
recom_result[i,1] <- as.character(subset(movies,
movies$movieId == as.integer(recom_list[[1]][i]))$title)
}
colnames(recom_result) <- "User-Based Collaborative Filtering Recommended Titles"
return(recom_result)
}
}
48:2 (Top Level) R Script
Console ~/Documents/movieRec-master/
recom_result <- data.frame(matrix(NA,10))
if (as.character(recom_list[1])=='character(0)'){
no_result[1,1] <- "Sorry, there is not enough information in our database on the movies you've selected. Try to se
lect different movies you like."
colnames(no_result) <- "No results"
return(no_result)
} else {
for (i in c(1:10)){
recom_result[i,1] <- as.character(subset(movies,
movies$movieId == as.integer(recom_list[[1]][i]))$title)
}
colnames(recom_result) <- "User-Based Collaborative Filtering Recommended Titles"
return(recom_result)
}
}
Environment History
Import Dataset List
Global Environment
Data
movies 9125 obs. of 3 variables
movies2 7298 obs. of 3 variables
ratings 105339 obs. of 4 variables
Functions
movie_recomme.. function (input, input2, input3)
Files Plots Packages Help Viewer
Zoom Export


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1 helpercode.R server.R
Reload App
#server.R
library(shiny) #Shiny. by RStudio. A web application framework for R.
library(proxy)#If your networking environment requires outbound network
#connections to go through a HTTP or HTTPS proxy, you should configure R to use the proxy for all downloads.
library(recommenderlab)
library(reshape2)#Reshape2 is a reboot of the reshape package.
source("helpercode.R")
search <- read.csv("search.csv", stringsAsFactors=FALSE)
search
ratings <- read.csv("ratings.csv", header = TRUE)
search <- search[-which((search$movieId %in% ratings$movieId) == FALSE),]
formatInput <- function(v,a,d){
This function formats the user's input of Valence-Arousal-Dominance
and outputs them as a vector
c(v,a,d)
219:3 (Top Level) R Script
Console ~/Documents/movieRec-master/
lect different movies you like."
colnames(no_result) <- "No results"
return(no_result)
} else {
for (i in c(1:10)){
recom_result[i,1] <- as.character(subset(movies,
movies$movieId == as.integer(recom_list[[1]][i]))$title)
}
colnames(recom_result) <- "User-Based Collaborative Filtering Recommended Titles"
return(recom_result)
}
}
> runApp()
Listening on http://127.0.0.1:6802
Environment History
Import Dataset List
Global Environment
Data
movies 9125 obs. of 3 variables
movies2 7298 obs. of 3 variables
ratings 105339 obs. of 4 variables
Functions
movie_recomme.. function (input, input2, input3)
Files Plots Packages Help Viewer
Zoom Export

CHAPTER -4

Testing

4.1 Test Cases , Inputs and Outputs



Select the Genres of movie.Select most rated movie of selected genres. Such That intersections of genres of three selected movie should not be null.

Output = Movies of Genre1∩ Movies of Genre2∩ Movie of Genre3.

CHAPTER -5

Application

- Product Recommendations: Perhaps the most important use of recommendation systems is at on-line retailers. We have noted how Amazonor similar on-line vendors strive to present each returning user with some
- Movie Recommendations: Netflix offers its customers recommendations of movies they might like. These recommendations are based on ratings provided by users, much like the ratings suggested in the example utility matrix. The importance of predicting ratings accurately is so high, that Netflix offered a prize of one million dollars for the first algorithm that could beat its own recommendation system by 10%.1Theprize was finally won in 2014, by a team of researchers called "Bellkor's Pragmatic Chaos," after over three years of competition.

CHAPTER -6

Conclusion and Strengths/weaknesses Discussion

In this project, I have developed and evaluated a collaborative filtering recommender (CFR) system for recommending movies. The online app was created to demonstrate the User-based Collaborative Filtering approach for recommendation model.

Let's discuss the strengths and weaknesses of the User-based Collaborative Filtering approach in general.

Strengths: User-based Collaborative Filtering gives recommendations that can be complements to the item the user was interacting with. This might be a stronger recommendation than what a item-based recommender can provide as users might not be looking for direct substitutes to a movie they had just viewed or previously watched.

Weaknesses: User-based Collaborative Filtering is a type of Memory-based Collaborative Filtering that uses all user data in the database to create recommendations. Comparing the pairwise correlation of every user in your dataset is not scalable. If there were millions of users, this computation would be very time consuming. Possible ways to get around this would be to implement some form of dimensionality reduction, such as Principal Component Analysis, or to use a model-based algorithm instead. Also, user-based collaborative filtering relies on past user choices to make future recommendations. The implications of this is that it assumes that a user's taste and preference remains more or less constant over time, which might not be true and makes it difficult to pre-compute user similarities offline.

CHAPTER -7

References

1.https://rpubs.com/jeknov/movieRec

2.https://www.udacity.com/course/data-analysis-with-r--ud651

3.www.wikipedia.com

4.mit ocw