MovieLens Recommender System

CWID	First name	Last Name	IIT Email
A20401921	Anusha	Satish	athattehalli@hawk.iit.edu
A20436206	Sivaranjani	Prabasankar	sprabasankar@hawk.iit.edu

Table of Contents

1. Introduction

In the past, people used to shop in a physical store, in which the items available are limited. For instance, the number of movies that can be placed in a Blockbuster store depends on the size of that store. By contrast, nowadays, the Internet allows people to access abundant resources online. For example, Netflix has an enormous collection of movies. Although the amount of available information increased, a new problem arose as user had a hard time selecting the items they want to see. This is where the recommender system comes in. Moreover, using recommendations users don't need to waste time and money for a low rated movie.

So, we want to make a recommender system which will help user to overcome the problem of selecting movies, as well as for websites in suggesting movies related to user genre using their history.

2. Data

The dataset has metadata for 40+k movies released on or before July 2017 and listed in Full MovieLens. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages. This dataset also has files containing 26 million ratings from 270,000 users and ratings are on a scale of 1-5 and have been obtained from the official Group Lens website. This dataset has been retrieved from the link: https://www.kaggle.com/rounakbanik/the-movies-dataset

To work on this project, we have chosen the dataset provided by Rounak Banik which consists of the following files:

keywords.csv: Contains the movie plot keywords for MovieLens movies.

credits.csv: Consists of Cast and Crew Information for all the movies.

links.csv: The file that contains the TMDB and IMDB IDs of all the movies featured.

Rating.csv

User ID ➔ Unique Id assigned to each user

Movie ID ➔ Unique Id assigned to each movie

Rating ➔ Ratings made on a 5-star scale, with half-star increments

Timestamp ➔ Represent seconds since midnight Coordinated Universal Time (UTC) of 1/1/1970.

```
> str(Movie_ratings)
'data.frame':   100004 obs. of  4 variables:
 $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ movieId  : int  31 1029 1061 1129 1172 1263 1287 1293 1339 1343 ...
 $ rating   : num  2.5 3 3 2 4 2 2 2 3.5 2 ...
 $ timestamp: int  1260759144 1260759179 1260759182 1260759185 1260759205
 1260759151 1260759187 1260759148 1260759125 1260759131 ...
>
```

```
> head(Movie_ratings)
  userId movieId rating  timestamp
1      1      31    2.5 1260759144
2      1    1029    3.0 1260759179
3      1    1061    3.0 1260759182
4      1    1129    2.0 1260759185
5      1    1172    4.0 1260759205
6      1    1263    2.0 1260759151
>
```

moviedata.csv: The main Movies Metadata file.

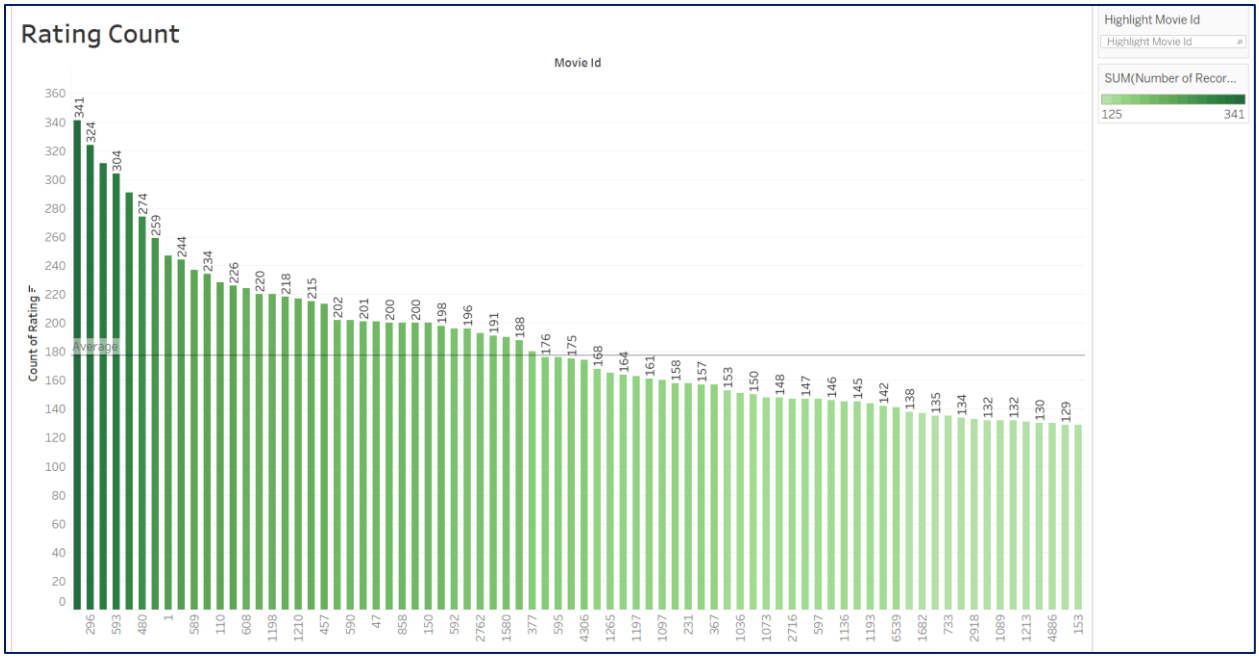
```
> head(Movie_data)
  movieId                               title                                        genres
1       1                    Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
2       2                      Jumanji (1995)                  Adventure|Children|Fantasy
3       3             Grumpier Old Men (1995)                              Comedy|Romance
4       4            Waiting to Exhale (1995)                        Comedy|Drama|Romance
5       5 Father of the Bride Part II (1995)                                      Comedy
6       6                            Heat (1995)                       Action|Crime|Thriller
>
```

```
> str(Movie_data)
'data.frame':   27278 obs. of  3 variables:
 $ movieId: int  1 2 3 4 5 6 7 8 9 10 ...
 $ title  : Factor w/ 27262 levels "'71 (2014)","'burbs, The (1989)",..: 2
4769 12877 10245 25847 8316 10735 20392 24615 22731 9887 ...
 $ genres : Factor w/ 1342 levels "(no genres listed)",..: 478 565 1029 96
9 894 351 1029 541 2 175 ...
>
```

IMDBData.csv: The main Movies Metadata file.

	fn	tid	title	wordsInTi	url	imdbRatin	ratingCou	duration	year	type	nrOfWins	nrOfNomi	nrOfPhotc	nrOfNews	nrOfUserF	nrOfGenre	Action	Adult	Adventure	Animation	Bic
2	titles01/tti	tt0012349	Der Vagab	der vagab	http://ww	8.4	40550	3240	1921	video	1	0	19	96	85	3	0	0	0	0	
3	titles01/tti	tt0015864	Goldrausc	goldrausc	http://ww	8.3	45319	5700	1925	video	2	1	35	110	122	3	0	0	1	0	
4	titles01/tti	tt0017136	Metropoli	metropoli	http://ww	8.4	81007	9180	1927	video	3	4	67	428	376	2	0	0	0	0	
5	titles01/tti	tt0017925	Der Gener	der gener	http://ww	8.3	37521	6420	1926	video	1	1	53	123	219	3	1	0	1	0	
6	titles01/tti	tt0021749	Lichter de	lichter der	http://ww	8.7	70057	5220	1931	video	2	0	38	187	186	3	0	0	0	0	
7	titles01/tti	tt0022100	M (1931)	m	http://ww	8.5	73726	7020	1931	video	1	0	28	4	254	3	0	0	0	0	
8	titles01/tti	tt0025316	Es geschal	es geschal	http://ww	8.3	46503	6300	1934	video	4	1	40	183	211	2	0	0	0	0	
9	titles01/tti	tt0027977	Moderne :	moderne :	http://ww	8.6	90847	5220	1936	video	3	1	44	27	180	2	0	0	0	0	
10	titles01/tti	tt0031381	Vom Winc	vom wind	http://ww	8.2	160414	14280	1939	video	10	6	143	1263	653	3	0	0	0	0	
11	titles01/tti	tt0031679	Mr. Smith	mr smith	http://ww	8.4	58169	7740	1939	video	4	10	34	110	226	1	0	0	0	0	
12	titles01/tti	tt0032138	Der Zaube	der zaube	http://ww	8.1	209506	6120	1939	video	6	12	126	2363	477	3	0	0	1	0	
13	titles01/tti	tt0032551	FrÃ¼chte	fr chte de	http://ww	8.2	45737	7740	1940	video	6	5	20	135	257	1	0	0	0	0	
14	titles01/tti	tt0032553	Der groÃŸ	der gro e	http://ww	8.5	87969	7500	1940	video	5	1	37	181	173	3	0	0	0	0	
15	titles01/tti	tt0032976	Rebecca (:	rebecca	http://ww	8.3	65670	7800	1940	video	2	9	47	328	241	3	0	0	0	0	
16	titles01/tti	tt0033467	Citizen Kar	citizen kar	http://ww	8.5	228617	7140	1941	video	7	10	103	1439	1101	2	0	0	0	0	
17	titles01/tti	tt0033870	Die Spur d	die spur d	http://ww	8.2	86012	6000	1941	video	1	0	67	332	294	3	0	0	0	0	
18	titles01/tti	tt0034583	Casablanc	casablanc	http://ww	8.6	296802	6120	1942	video	5	6	153	382	960	3	0	0	0	0	
19	titles01/tti	tt0036775	Frau ohne	frau ohne	http://ww	8.5	71266	6420	1944	video	3	0	43	231	260	3	0	0	0	0	
20	titles01/tti	tt0036868	Die bester	die besten	http://ww	8.3	30002	10320	1946	video	13	1	15	129	218	3	0	0	0	0	
21	titles01/tti	tt0038355	Tote schla	tote schla	http://ww	8.1	50020	6840	1946	video	1	0	52	171	215	3	0	0	0	0	

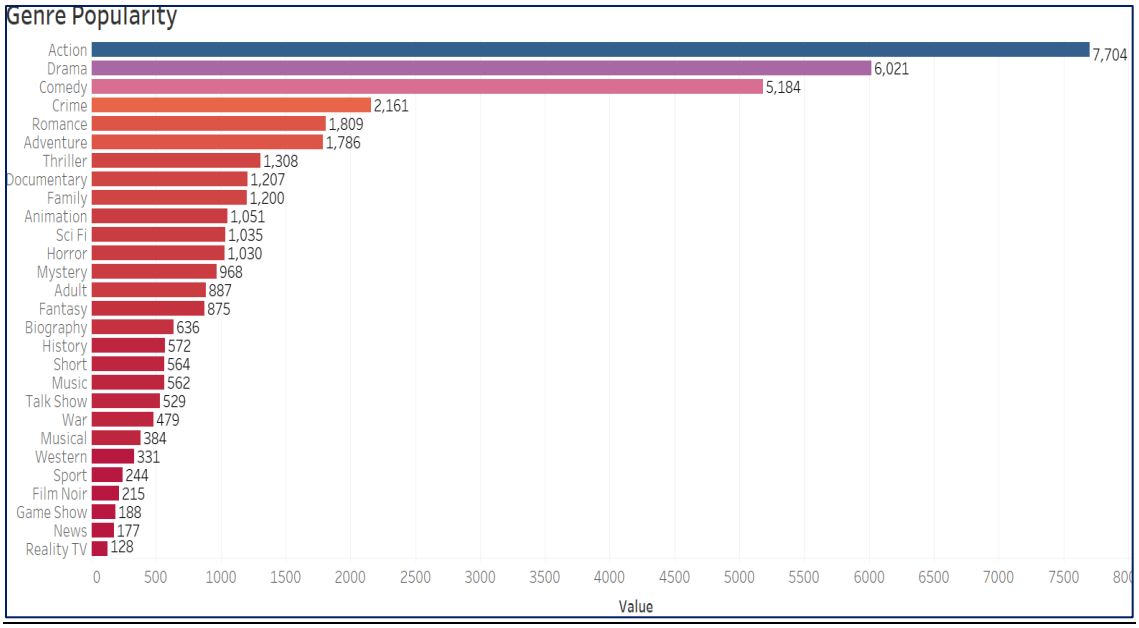
Most Rated Movies



Top rated Movies

	title	avg_rating	rating_cnt
	<fct>	<dbl>	<int>
1	Godfather, The (1972)	4.49	200
2	Shawshank Redemption, The (1994)	4.49	311
3	Godfather: Part II, The (1974)	4.39	135
4	Usual Suspects, The (1995)	4.37	201
5	Schindler's List (1993)	4.30	244
6	One Flew Over the Cuckoo's Nest (1975)	4.26	144
7	Fargo (1996)	4.26	224
8	Pulp Fiction (1994)	4.26	324
9	American Beauty (1999)	4.24	220
10	Dark Knight, The (2008)	4.24	121
11	Casablanca (1942)	4.24	117
12	Star Wars: Episode V - The Empire Strikes Back (1980)	4.23	234
13	Memento (2000)	4.23	132
14	Taxi Driver (1976)	4.22	118
15	Monty Python and the Holy Grail (1975)	4.22	145
16	Star Wars: Episode IV - A New Hope (1977)	4.22	291
17	Dr. Strangelove or: How I Learned to Stop Worrying and …	4.21	105
18	Princess Bride, The (1987)	4.21	163
19	Goodfellas (1990)	4.20	131
20	Raiders of the Lost Ark (Indiana Jones and the Raiders …	4.19	220

Genre Popularity



3. Problem to be Solved

- Movie websites must help users to take correct decision in choosing movies. Eventually it will increase their sales and improve user web browsing experience. Due to Information overload movie website faces problem while providing personalized data to their users.

- We have columns represented in both categorical and numerical features. We need to confirm if all the predictors have a significant effect on the Ratings.

- There are some missing and irrelevant values in some features which must be resolved before processing data.

- We are expecting many multicollinearity problems as some features are inter-related with each other.

4. Solution

To address the above problems, we wish to work towards achievement of following solutions:

1. **Recommender Systems** helps website in suggesting personalized recommendations for each user. There are different types of recommender systems such as

   I. User Based Collaborative Filtering – To find the similarity between target users and other users.
   II. Item Based Collaborative Filtering – To measure the similarity between the items that the target user rates with other items.

III. Content Based Recommender System– To analyze an item a user interacted with and recommend that are like the item. By using this approach, we can recommend movies that are of highest similarity based on criteria's like Genre preferred by the user, Movies of same Actor/ Director.

IV. Matrix Factorization –To decompose higher dimension to lower dimension matrices by considering latent factors for movie and users respectively.

2. **Classification model**

We plan to build multiple classification models using SVM and KNN to predict the rating of a new movie, when added. We compare the accuracies of multiple models to find the best classification model among them.

3. **Feature selection and Feature Reduction**

We apply feature selection and reduction techniques to remove less likely or multicollinear features for our predictions.

5. Experiments and Results

5.1 Methods and Process

5.1.1 Classification

I) K-Nearest Neighbor

Step 1: Loading Data in R

```
#-------------------------------------------------------------------------------
# Classification
#-------------------------------------------------------------------------------

#Getting present working Directory
getwd()

#Modifying working directory
setwd('D:/Fall-19/Data Mining/Project/Movielens_Project/Classification/')

# Loading data from CSV file
Movie_IMDB=read.csv('imdb.csv',header=T)
str(Movie_IMDB)
# Checking for missing values
nrow(is.na(Movie_IMDB))

# installing required packages for SVM
#install.packages('caret', dependencies = TRUE)
library(caret)
```

Step 2: Data preprocessing - Handling Dirty values

```
#--------------------------Data pre processing-----------------------------------------
# Handling dirty values for No.of.Wins column

Movie_IMDB$nrOfWins=as.numeric(as.character(Movie_IMDB$nrOfWins))
#wins_IMDB=ifelse(is.na(wins_IMDB),0,wins_IMDB)
summary(Movie_IMDB$nrOfWins)

# Handling dirty values for No.of.Nominations column
Movie_IMDB$nrOfNominations=as.numeric(as.character(Movie_IMDB$nrOfNominations))
summary(Movie_IMDB$nrOfNominations)

# Handling dirty values for Rating count column
Movie_IMDB$ratingCount=as.numeric(as.character(Movie_IMDB$ratingCount))
summary(Movie_IMDB$ratingCount)

# Handling dirty values for NewsArticles count
Movie_IMDB$nrOfNewsArticles=as.numeric(as.character(Movie_IMDB$nrOfNewsArticles))
summary(Movie_IMDB$nrOfNewsArticles)

# Genre split
Movie_IMDB$Action=as.numeric(as.character(Movie_IMDB$Action))
summary(Movie_IMDB$Action)

Movie_IMDB$Adult=as.numeric(as.character(Movie_IMDB$Adult))
summary(Movie_IMDB$Adult)
```

Step 3: Converting Dependent variable to categorical

```
# Checking dirty values for other features
summary(Movie_IMDB$nrOfUserReviews)
summary(Movie_IMDB$nrOfGenre)

Movie_IMDB$nrOfPhotos=as.numeric(as.character(Movie_IMDB$nrOfPhotos))
summary(Movie_IMDB$nrOfPhotos)

Movie_IMDB$duration=as.numeric(as.character(Movie_IMDB$duration))
summary(Movie_IMDB$duration)

# Handling dirty values for Rating IMDB
Movie_IMDB$imdbRating=as.numeric(as.character(Movie_IMDB$imdbRating))
summary(Movie_IMDB$imdbRating)

#--------------------------Data Transformation-----------------------------------------
Movie_IMDB$imdbRating=cut(Movie_IMDB$imdbRating,breaks = c(0.0,2.5,5.0,8.5,10.0),labels=c("Bad","Fair","Good","Amazing"),right=FALSE)
head(Movie_IMDB$imdbRating)
summary(Movie_IMDB)

#install.packages('caret', dependencies = TRUE)
library(caret)
```

Step 4: Building models

Model 1:

```
#--------------------------# Model 1 - Full MODEL-----------------------------------------

subset_1=c("imdbRating","nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","nrOfUserReviews","nrOfGen

IMDB_M1_1=Movie_IMDB[subset_1]

IMDB_M1_2=na.omit(IMDB_M1_1)
y1=IMDB_M1_2$imdbRating
head(IMDB_M1_2)

IMDB_M1_2=IMDB_M1_2[-1]

# Normalizing independent attributes
num_vars=sapply(IMDB_M1_2,is.numeric)
IMDB_M1_2[num_vars]=lapply(IMDB_M1_2[num_vars],scale)
head(IMDB_M1_2)
x1=IMDB_M1_2

KNN_M1=train(x1,y1,'knn',trControl = trainControl(method = 'cv',number = 10),tuneGrid = expand.grid(k=51:59))
print(KNN_M1)
```

Step 5: Feature Selection

```
#---------------------------Feature selection-------------------------------------------
control=rfeControl(functions = rfFuncs,method = "cv",number = 10)
Features_Select=rfe(IMDB_M3_2[,1:8],y3,sizes = c(1:8),rfeControl = control)
print(Features_Select)
predictors(Features_Select)
plot(Features_Select, type = c("g","o"))
```

Step 6: Building Models after Feature selection

Model 2:

```
#---------------------------# Model 2 - 8 Features MODEL-----------------------------------------

subset_2=c("imdbRating","nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","nrOfUserReviews","nrOfGen

IMDB_M2_1=Movie_IMDB[subset_2]
summary(IMDB_M2_1)

IMDB_M2_2=na.omit(IMDB_M2_1)
summary(IMDB_M2_2)

IMDB_M2_2=IMDB_M2_2[-1]
y2=IMDB_M2_2$imdbRating

# Normalizing independent attributes
num_vars=sapply(IMDB_M2_2,is.numeric)
IMDB_M2_2[num_vars]=lapply(IMDB_M2_2[num_vars],scale)
head(IMDB_M2_2)

x2=IMDB_M2_2

KNN_M2=train(x2,y2,'knn',trControl = trainControl(method = 'cv',number = 10),tuneGrid = expand.grid(k=51:59))
print(KNN_M2)
```

Model 3:

```
#---------------------------# Model 3 -Features selected MODEL-----------------------------------------

subset_3=c("imdbRating","nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","duration")

IMDB_M3_1=Movie_IMDB[subset_3]
head(IMDB_M3_1)

#IMDB_Data_2=Movie_IMDB[subset_1]
IMDB_M3_2=na.omit(IMDB_M3_1)
summary(IMDB_M3_2)

y3=IMDB_M3_2$imdbRating
IMDB_M3_2=IMDB_M3_2[-1]
# Normalizing independent attributes
num_vars=sapply(IMDB_M3_2,is.numeric)
IMDB_M3_2[num_vars]=lapply(IMDB_M3_2[num_vars],scale)
head(IMDB_M3_2)
summary(IMDB_M3_2)

# Normalizing independent attributes
num_vars=sapply(IMDB_M3_2,is.numeric)
IMDB_M3_2[num_vars]=lapply(IMDB_M3_2[num_vars],scale)
head(IMDB_M3_2)

x3=IMDB_M3_2
KNN_M3=train(x3,y3,'knn',trControl = trainControl(method = 'cv',number = 10),tuneGrid = expand.grid(k=51:59))
print(KNN_M3)
```

Model 4: (Excluding rating count)

```
#--------------------------# Model 4 -Features selected MODEL excluding popularity--------------------------

subset_4=c("imdbRating","nrOfWins","nrOfNominations","duration")

IMDB_M4_1=Movie_IMDB[subset_4]
head(IMDB_M4_1)

#IMDB_Data_2=Movie_IMDB[subset_1]
IMDB_M4_2=na.omit(IMDB_M4_1)
summary(IMDB_M4_2)

y4=IMDB_M4_2$imdbRating
IMDB_M4_2=IMDB_M4_2[-1]
# Normalizing independent attributes
num_vars=sapply(IMDB_M4_2,is.numeric)
IMDB_M4_2[num_vars]=lapply(IMDB_M4_2[num_vars],scale)
head(IMDB_M4_2)
summary(IMDB_M4_2)

# Normalizing independent attributes
num_vars=sapply(IMDB_M4_2,is.numeric)
IMDB_M4_2[num_vars]=lapply(IMDB_M4_2[num_vars],scale)
head(IMDB_M4_2)

x4=IMDB_M4_2
KNN_M4=train(x4,y4,'knn',trControl = trainControl(method = 'cv',number = 10),tuneGrid = expand.grid(k=51:59))
print(KNN_M4)
```

Model 5: (Excluding rating count, no. of. nomination)

```
#--------------------------# Model 5 -major features MODEL excluding popularity----------------------------
subset_5=c("imdbRating","nrOfGenre","nrOfPhotos","duration")

IMDB_M5_1=Movie_IMDB[subset_5]
head(IMDB_M5_1)

#IMDB_Data_2=Movie_IMDB[subset_1]
IMDB_M5_2=na.omit(IMDB_M5_1)
summary(IMDB_M5_2)

y5=IMDB_M5_2$imdbRating
IMDB_M5_2=IMDB_M5_2[-1]
# Normalizing independent attributes
num_vars=sapply(IMDB_M5_2,is.numeric)
IMDB_M5_2[num_vars]=lapply(IMDB_M5_2[num_vars],scale)
head(IMDB_M5_2)
summary(IMDB_M5_2)

# Normalizing independent attributes
num_vars=sapply(IMDB_M5_2,is.numeric)
IMDB_M5_2[num_vars]=lapply(IMDB_M5_2[num_vars],scale)
head(IMDB_M5_2)

x5=IMDB_M5_2
KNN_M5=train(x5,y5,'knn',trControl = trainControl(method = 'cv',number = 10),tuneGrid = expand.grid(k=51:59))
print(KNN_M5)
```

Step 7: Evaluating models

Model	Predictors	Accuracy
Model 1	All 36 features	0.9106010
Model 2	imdbRating, nrOfWins, nrOfNominations, ratingCount, nrOfNewsArticles, nrOfUserReviews, nrOfGenre, nrOfPhotos, duration	0.9112457
Model 3	imdbRating, nrOfWins, nrOfNominations, ratingCount, nrOfNewsArticles, duration	0.9107619
Model 4	imdbRating, nrOfWins, nrOfNominations, duration	0.9106008
Model 5	imdbRating, nrOfGenre, nrOfPhotos, duration	0.9106007

II) Support Vector Machine

Step 1: Loading Data in R

```
#-------------------------------------------------------------------------------
# Classification
#-------------------------------------------------------------------------------

#Getting present working Directory
getwd()

#Modifying working directory
setwd('D:/Fall-19/Data Mining/Project/Movielens_Project/Classification/')

# Loading data from CSV file
Movie_IMDB=read.csv('imdb.csv',header=T)
str(Movie_IMDB)
# Checking for missing values
nrow(is.na(Movie_IMDB))

# installing required packages for SVM
#install.packages('caret', dependencies = TRUE)
library(caret)
```

Step 2: Data preprocessing - Handling Dirty values

```
#--------------------------Data pre processing-----------------------------------------
# Handling dirty values for No.of.Wins column

Movie_IMDB$nrOfWins=as.numeric(as.character(Movie_IMDB$nrOfWins))
#wins_IMDB=ifelse(is.na(wins_IMDB),0,wins_IMDB)
summary(Movie_IMDB$nrOfWins)

# Handling dirty values for No.of.Nominations column
Movie_IMDB$nrOfNominations=as.numeric(as.character(Movie_IMDB$nrOfNominations))
summary(Movie_IMDB$nrOfNominations)

# Handling dirty values for Rating count column
Movie_IMDB$ratingCount=as.numeric(as.character(Movie_IMDB$ratingCount))
summary(Movie_IMDB$ratingCount)

# Handling dirty values for NewsArticles count
Movie_IMDB$nrOfNewsArticles=as.numeric(as.character(Movie_IMDB$nrOfNewsArticles))
summary(Movie_IMDB$nrOfNewsArticles)

# Genre split
Movie_IMDB$Action=as.numeric(as.character(Movie_IMDB$Action))
summary(Movie_IMDB$Action)

Movie_IMDB$Adult=as.numeric(as.character(Movie_IMDB$Adult))
summary(Movie_IMDB$Adult)
```

Step 3: Converting Dependent variable to categorical

```
# Checking dirty values for other features
summary(Movie_IMDB$nrOfUserReviews)
summary(Movie_IMDB$nrOfGenre)

Movie_IMDB$nrOfPhotos=as.numeric(as.character(Movie_IMDB$nrOfPhotos))
summary(Movie_IMDB$nrOfPhotos)

Movie_IMDB$duration=as.numeric(as.character(Movie_IMDB$duration))
summary(Movie_IMDB$duration)

# Handling dirty values for Rating IMDB
Movie_IMDB$imdbRating=as.numeric(as.character(Movie_IMDB$imdbRating))
summary(Movie_IMDB$imdbRating)

#--------------------------Data Transformation-----------------------------------------
Movie_IMDB$imdbRating=cut(Movie_IMDB$imdbRating,breaks = c(0.0,2.5,5.0,8.5,10.0),labels=c("Bad","Fair","Good","Amazing"),right=FALSE)
head(Movie_IMDB$imdbRating)
summary(Movie_IMDB)
```

Step 4: Building models

Model 1:

```
# Model 1
subset_1=c("nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","nrOfUserReviews","nrOfGenre","imdbRating")

subset_1=c("imdbRating","nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","nrOfUserReviews","nrOfGenre","nrOf

IMDB_Data_1=Movie_IMDB[subset_1]
summary(IMDB_Data_1)

IMDB_Data_2=na.omit(IMDB_Data_1)
head(IMDB_Data_2)
summary(IMDB_Data_2)

sum(is.na(IMDB_Data_2))
head(x1)
x1=IMDB_Data_2[subset_1]
x1=IMDB_Data_2[-1]
x1=IMDB_Data_2[-7]
y1=IMDB_Data_2$imdbRating
sum(is.na(y1))

dim(x1)
dim(y1)
# Executing SVM for dependent and independent attributes
# Implemented N(5) Fold cross validation of
SVM_M1 = train(x1,y1, data = IMDB_Data_2, method = "svmLinear", trControl=trainControl(method = "cv", number = 5), tuneLe
print(SVM_M1)
```

Step 5: Feature Selection

```
#---------------------------Feature selection-------------------------------------------
control=rfeControl(functions = rfFuncs,method = "cv",number = 10)
Features_Select=rfe(IMDB_M3_2[,1:8],y3,sizes = c(1:8),rfeControl = control)
print(Features_Select)
predictors(Features_Select)
plot(Features_Select, type = c("g","o"))
```

Step 6: Building Models after feature selection

Model 2:

```
#---------------------------# Model 2 - 8 Features MODEL----------------------------------------

subset_2=c("imdbRating","nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","nrOfUserReviews","nrOfGen

IMDB_M2_1=Movie_IMDB[subset_2]
summary(IMDB_M2_1)

IMDB_M2_2=na.omit(IMDB_M2_1)
summary(IMDB_M2_2)

IMDB_M2_2=IMDB_M2_2[-1]
y2=IMDB_M2_2$imdbRating

# Normalizing independent attributes
num_vars=sapply(IMDB_M2_2,is.numeric)
IMDB_M2_2[num_vars]=lapply(IMDB_M2_2[num_vars],scale)
head(IMDB_M2_2)

x2=IMDB_M2_2

KNN_M2=train(x2,y2,'knn',trControl = trainControl(method = 'cv',number = 10),tuneGrid = expand.grid(k=51:59))
print(KNN_M2)
```

Model 3:

```
#--------------------------# Model 3 -Features selected MODEL----------------------------------------

subset_3=c("imdbRating","nrOfWins","nrOfNominations","ratingCount","nrOfNewsArticles","duration")

IMDB_M3_1=Movie_IMDB[subset_3]
head(IMDB_M3_2)

#IMDB_Data_2=Movie_IMDB[subset_1]
IMDB_M3_2=na.omit(IMDB_M3_1)
summary(IMDB_M3_2)

y3=IMDB_M3_2$imdbRating
IMDB_M3_2=IMDB_M3_2[-1]
head(IMDB_M3_2)
summary(IMDB_M3_2)

x3=IMDB_M3_2

SVM_M3 = train(x3,y3, data = IMDB_M3_2, method = "svmLinear", trControl=trainControl(method = "cv", number = 10), tuneLeng
print(SVM_M3)
```

Model 4: (Excluding rating count)

```
#--------------------------# Model 4 -Features selected MODEL excluding popularity-------------------------------------

subset_4=c("imdbRating","nrOfWins","nrOfNominations","duration")

IMDB_M4_1=Movie_IMDB[subset_4]
head(IMDB_M4_1)

#IMDB_Data_2=Movie_IMDB[subset_1]
IMDB_M4_2=na.omit(IMDB_M4_1)
summary(IMDB_M4_2)

y4=IMDB_M4_2$imdbRating
IMDB_M4_2=IMDB_M4_2[-1]
head(IMDB_M4_2)
summary(IMDB_M4_2)

x4=IMDB_M4_2

SVM_M4 = train(x4,y4, data = IMDB_M4_2, method = "svmLinear", trControl=trainControl(method = "cv", number = 10), tuneLeng
print(SVM_M4)
```

Model 5: (Excluding rating count, no. of. nomination)

```
#--------------------------# Model 5 -Major Features MODEL excluding popularity----------------------------------------

subset_5=c("imdbRating","nrOfGenre","nrOfPhotos","duration")

IMDB_M5_1=Movie_IMDB[subset_5]
head(IMDB_M5_1)

#IMDB_Data_2=Movie_IMDB[subset_1]
IMDB_M5_2=na.omit(IMDB_M5_1)
summary(IMDB_M5_2)

y5=IMDB_M5_2$imdbRating
IMDB_M5_2=IMDB_M5_2[-1]
head(IMDB_M5_2)
summary(IMDB_M5_2)

x5=IMDB_M5_2

SVM_M5 = train(x5,y5, data = IMDB_M5_2, method = "svmLinear", trControl=trainControl(method = "cv", number = 10), tuneLeng
print(SVM_M5)
```

Step 7: Evaluating models

Model	Predictors	Accuracy
Model 1	All 36 features	0.9106007
Model 2	imdbRating, nrOfWins, nrOfNominations, ratingCount, nrOfNewsArticles, nrOfUserReviews, nrOfGenre, nrOfPhotos, duration	0.9112457
Model 3	imdbRating, nrOfWins, nrOfNominations, ratingCount, nrOfNewsArticles, duration	0.9106008
Model 4	imdbRating, nrOfWins, nrOfNominations, duration	0.9106008
Model 5	imdbRating, nrOfGenre, nrOfPhotos, duration	0.910601

Interpretation

We tried to predict rating group for IMDB movies using SVM and KNN. From the values we could see that there is no significant difference in accuracies between models. We ought to exclude some features which has significant impact in predicting rating group to avoid cold start problem. In future we would like to include more features from other data set for our analysis.

5.1.2 Clustering

OBJECTIVE: To find out similarities within groups of people in order to build a movie recommending system for users.

Step 1: Loading Data in R

Dataset stored in CSV format has been loaded in python using read_csv in Pandas library.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import csr_matrix
import io, os, sys, types
import import_ipynb
import helper
from helper import draw_clusters

# Import the Movies dataset
movies = pd.read_csv('movies_small.csv')
movies.head()

# Import the ratings dataset
ratings = pd.read_csv('ratings_small.csv')
ratings.head()



# Function to get the genre ratings
def get_genre_ratings(ratings, movies, genres, column_names):
    genre_ratings = pd.DataFrame()
    for genre in genres:
        genre_movies = movies[movies['genres'].str.contains(genre) ]
        avg_genre_votes_per_user = ratings[ratings['movieId'].isin(genre_movies['movieId'])].loc[:, ['userId', '

        genre_ratings = pd.concat([genre_ratings, avg_genre_votes_per_user], axis=1)

    genre_ratings.columns = column_names
    return genre_ratings
```

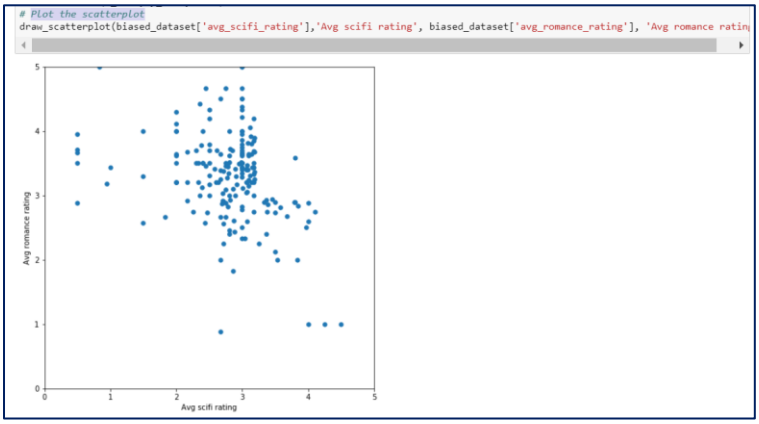
Step 2: Calculate the average rating of romance and scifi movies

```
# Function to get the genre ratings
def get_genre_ratings(ratings, movies, genres, column_names):
    genre_ratings = pd.DataFrame()
    for genre in genres:
        genre_movies = movies[movies['genres'].str.contains(genre) ]
        avg_genre_votes_per_user = ratings[ratings['movieId'].isin(genre_movies['movieId'])].loc[:, ['userId', 'rating']].groupb

        genre_ratings = pd.concat([genre_ratings, avg_genre_votes_per_user], axis=1)

    genre_ratings.columns = column_names
    return genre_ratings


# Calculate the average rating of romance and scifi movies
genre_ratings = get_genre_ratings(ratings, movies, ['Romance', 'Sci-Fi'], ['avg_romance_rating', 'avg_scifi_rating'])
genre_ratings.head()
```

	avg_romance_rating	avg_scifi_rating
1	3.50	2.40
2	3.59	3.80
3	3.69	3.17
4	4.52	4.31
5	4.16	3.67

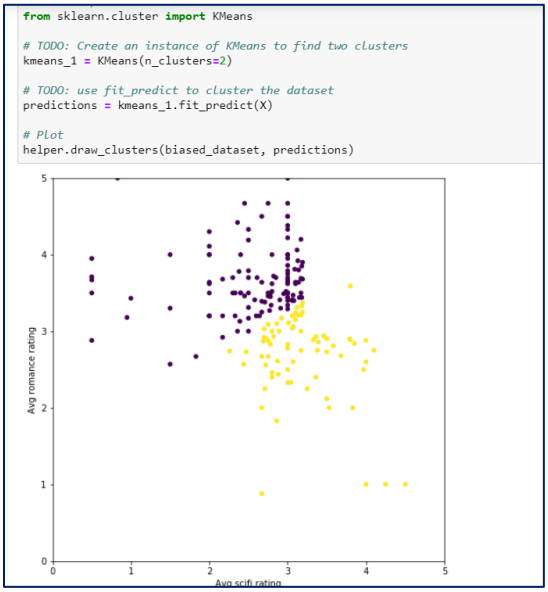
Step 3: Plot the scatterplot

```
# Plot the scatterplot
draw_scatterplot(biased_dataset['avg_scifi_rating'],'Avg scifi rating', biased_dataset['avg_romance_rating'], 'Avg romance ratin
```

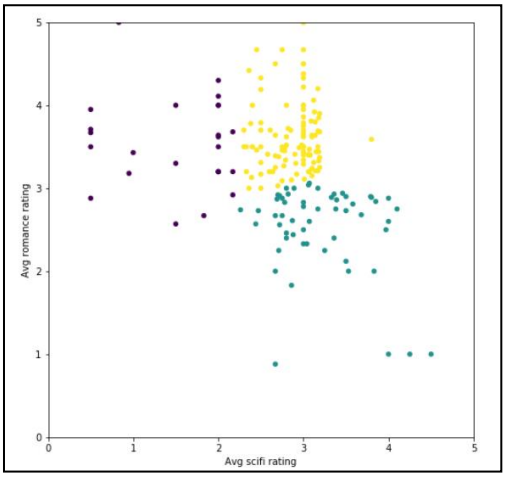


Step 4: Create 2 clusters and draw scatter plot

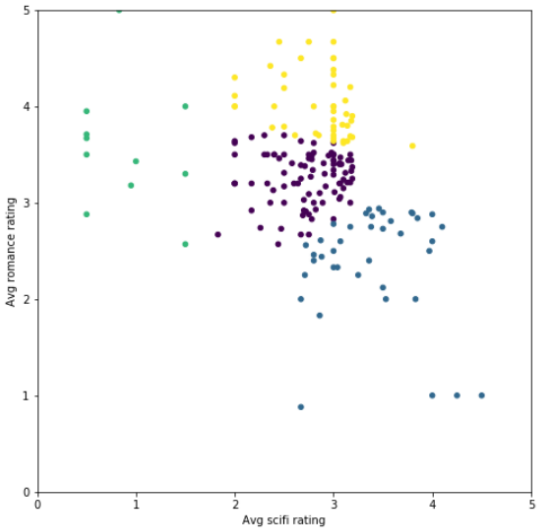
```
from sklearn.cluster import KMeans

# TODO: Create an instance of KMeans to find two clusters
kmeans_1 = KMeans(n_clusters=2)

# TODO: use fit_predict to cluster the dataset
predictions = kmeans_1.fit_predict(X)

# Plot
helper.draw_clusters(biased_dataset, predictions)
```



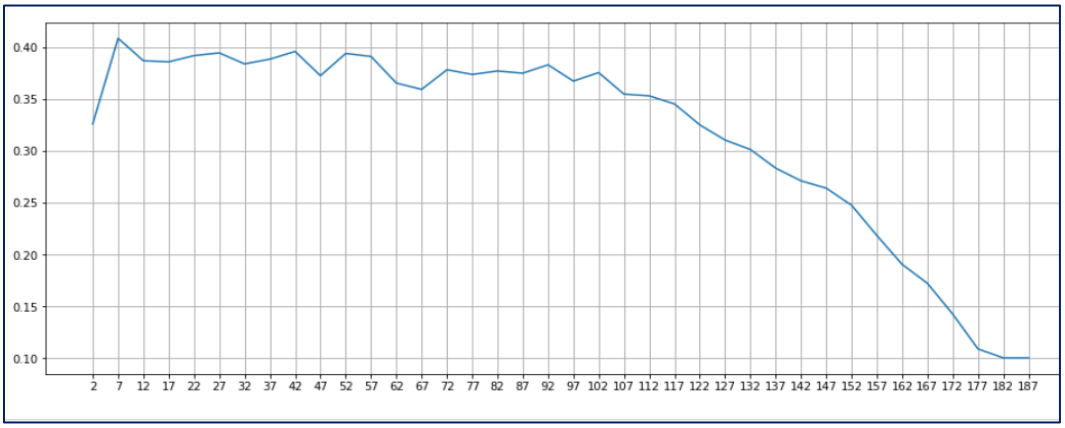
Step 5: Create 3 clusters and draw scatter plot



Step 6: Create 4 clusters and draw scatter plot

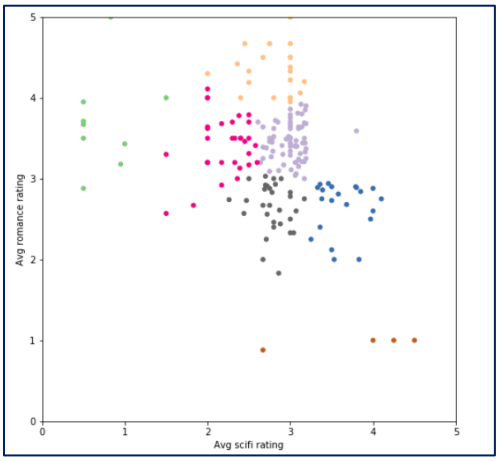


Step 7: Draw a graph to find ELBOW points



Step 8: Create 7 Clusters and draw scatter plot

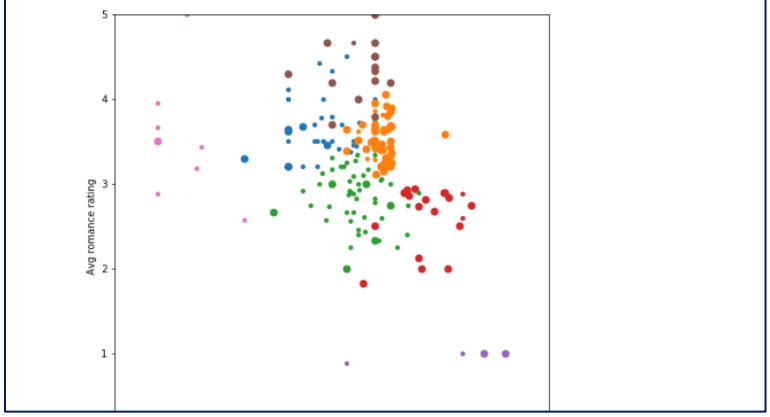
By using above graph, we found that 7 has good ELBOW point. Hence, we are creating 7 clusters.



Step 9: Adding another Genre for our analysis

We are using the x and y axes as romance and sci-fi ratings. In addition, we are plotting the size of the dot to represent the ratings of the action movies. (the bigger the dot the higher the action rating).

```
In [16]: X_with_action = biased_dataset_3_genres[['avg_scifi_rating',
                                                  'avg_romance_rating',
                                                  'avg_action_rating']].values

In [17]: # TODO: Create an instance of KMeans to find seven clusters
         kmeans_5 = KMeans(n_clusters=7)

         # TODO: use fit_predict to cluster the dataset
         predictions_5 = kmeans_5.fit_predict(X_with_action)

         # plot
         helper.draw_clusters_3d(biased_dataset_3_genres, predictions_5)
```



Step 10: Taking a bigger picture of the dataset

We will sort the dataset by the most rated movies and the users that have rated the greatest number of movies.

title	'Til There Was You (1997)	'burbs, The (1989)	1-900 (06) (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1996)	101 Dalmatians (One Hundred and One Dalmatians) (1961)	12 Angry Men (1957)	13th Warrior, The (1999)	187 (One Eight Seven) (1997)	1984 (Nineteen Eighty-Four) (1984)	2 Days in the Valley (1996)	2 ou 3 choses que je sais d'elle (2 or 3 Things I Know About Her) (1967)	20 Dates (1998)	20,000 Leagues Under the Sea (1954)	200 Cigarettes (1999)	2001: A Space Odyssey (1968)	Cc (
userId																	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
6	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
dataset dimensions:  (18, 30)
```

[19]:

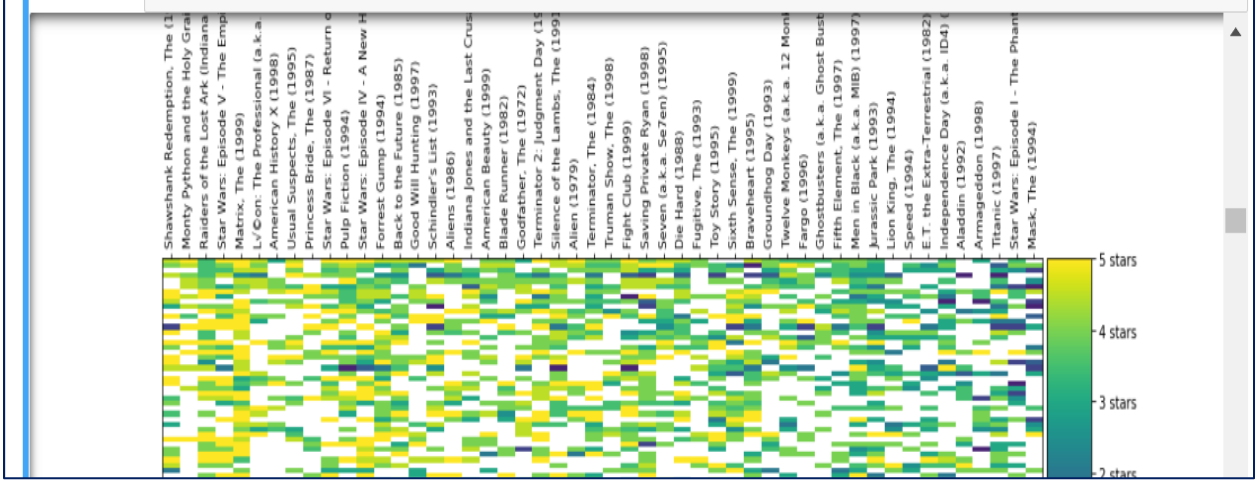
title	Forrest Gump (1994)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)	Star Wars: Episode IV - A New Hope (1977)	Jurassic Park (1993)	Matrix, The (1999)	Toy Story (1995)	Schindler's List (1993)	Terminator 2: Judgment Day (1991)	...	Dances with Wolves (1990)	Fight Club (1999)	Usual Suspects, The (1995)	Seven (a.k.a. Se7en) (1995)	Lion King, The (1994)	Godfat The (19
653	4.0	5.0	5.0	4.5	5.0	4.5	5.0	5.0	5.0	5.0	...	4.5	5.0	5.0	4.5	5.0	
29	5.0	5.0	5.0	4.0	4.0	4.0	3.0	4.0	5.0	4.0	...	5.0	4.0	5.0	4.0	3.0	
72	5.0	5.0	5.0	4.5	4.5	4.0	4.5	5.0	5.0	3.0	...	4.5	5.0	5.0	5.0	5.0	
508	4.0	5.0	4.0	4.0	5.0	3.0	4.5	3.0	5.0	2.0	...	5.0	4.0	5.0	4.0	3.5	
14	1.0	5.0	2.0	5.0	5.0	3.0	5.0	2.0	4.0	4.0	...	3.0	5.0	5.0	5.0	4.0	

5 rows × 30 columns

Step 10: Plotting heatmap

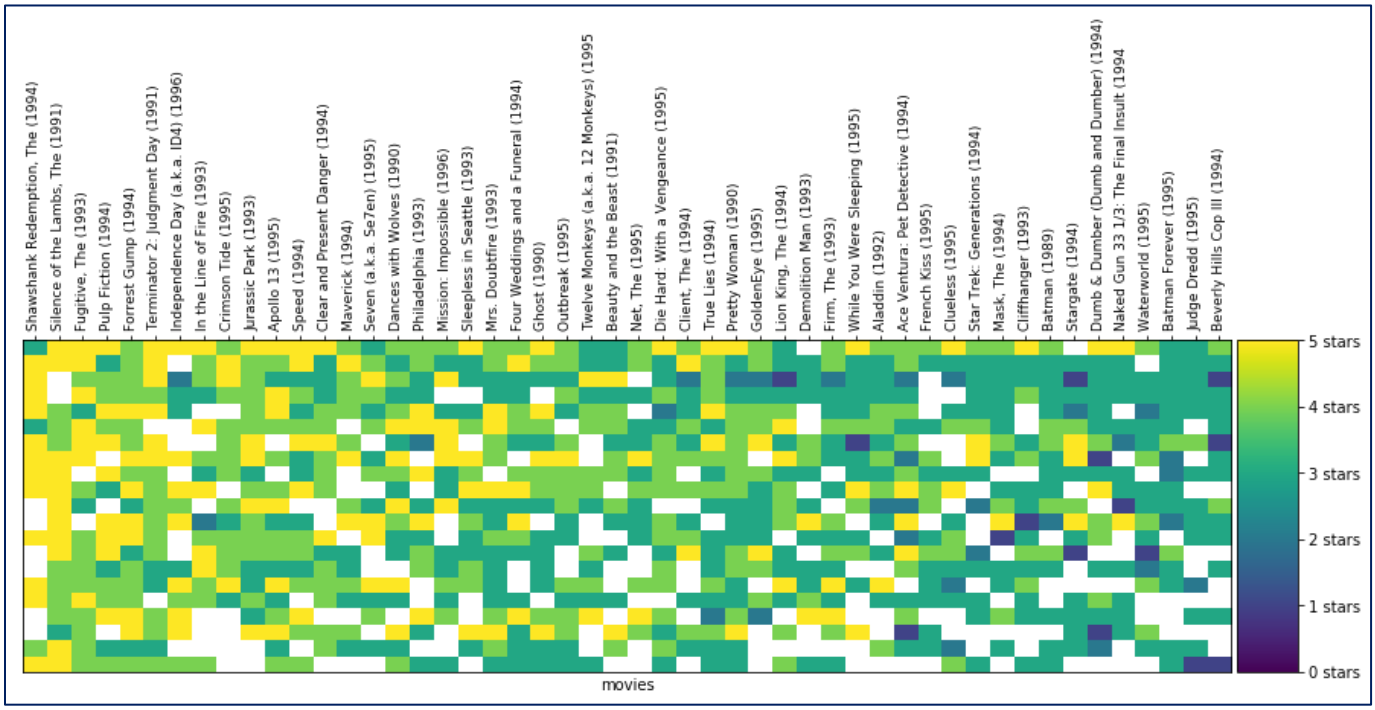
As we have a high number of dimensions and data to be plotted, the preferred visual is 'heatmaps'.

```
In [21]: user_movie_ratings =  pd.pivot_table(ratings_title, index='userId', columns= 'title', values='rating')
         most_rated_movies_1k = helper.get_most_rated_movies(user_movie_ratings, 1000)

In [22]: sparse_ratings = csr_matrix(pd.SparseDataFrame(most_rated_movies_1k).to_coo())

In [23]: # 20 clusters
         predictions = KMeans(n_clusters=20, algorithm='full').fit_predict(sparse_ratings)

In [24]: max_users = 70
         max_movies = 50

         clustered = pd.concat([most_rated_movies_1k.reset_index(), pd.DataFrame({'group':predictions})], axis=1)
         helper.draw_movie_clusters(clustered, max_users, max_movies)
```



Step 11: Heatmap

As we have a high number of dimensions and data to be plotted, we are choosing 'heatmaps'.



To understand the heatmap:

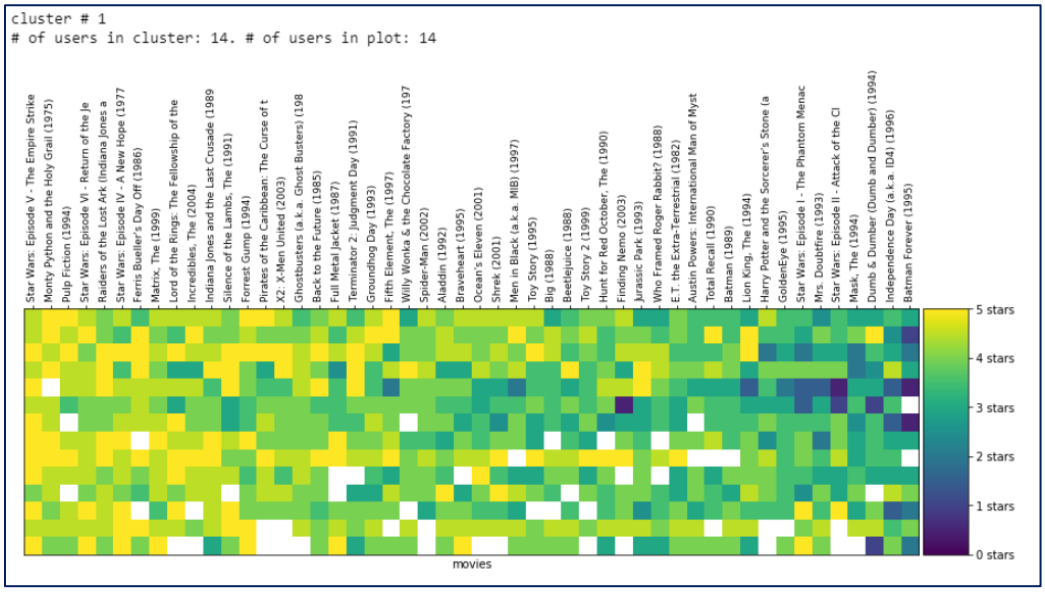
- Each column is a different movie.
- Each row is a different user.
- The cell's color is the rating that each user has given to each film. The values for each color can be checked in the scale of the right.
- The white values correspond to users that haven't rated the movie.

In order to improve the performance of the model, we'll only use ratings for 1000 movies.

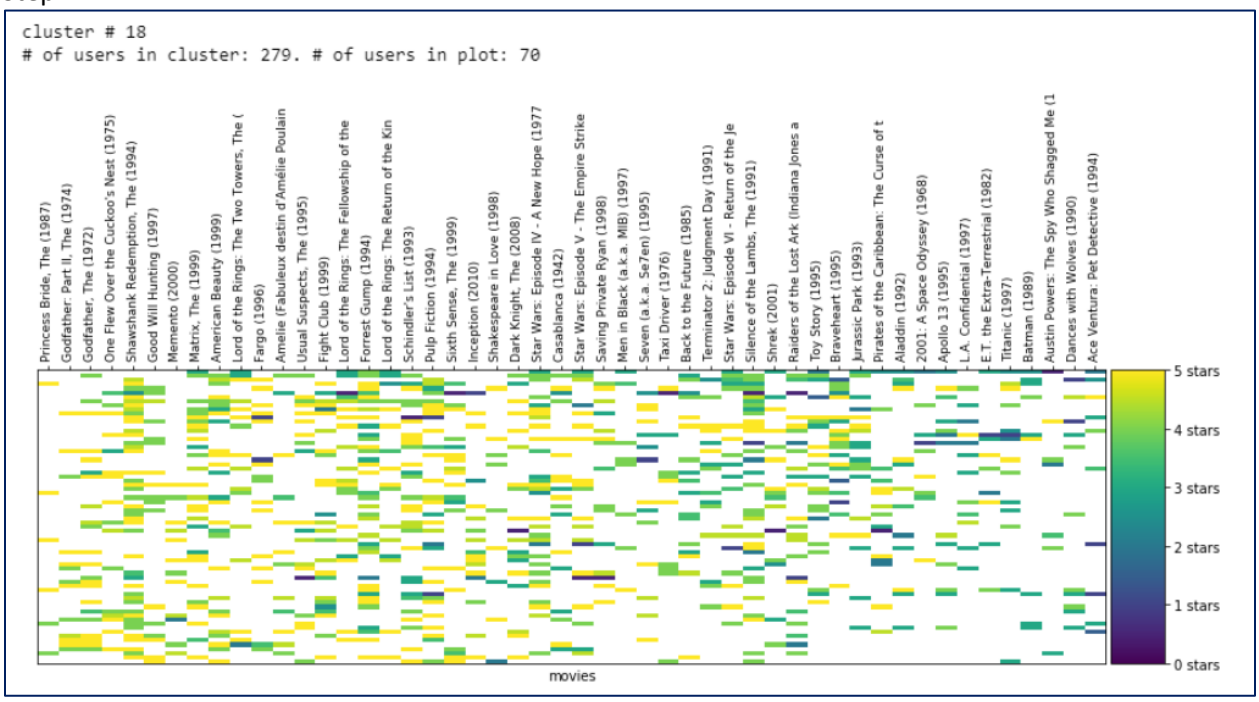
Step 12: Massive Clustering

Let us take an arbitrary number of clusters in order to make an analysis of the results obtained and spot certain trends and commonalities within each group. Let us take K = 20 and plot each cluster as a heatmap. Just displaying two clusters instead of all twenty.

Cluster 1



Step 12:



Things we can notice from these heatmaps:

- The more vertical lines of the same color in the cluster, the more similar the ratings will be in that cluster.

- Some clusters are sparser than others, that show that the algorithm tends to group also people that watch and rate less movies.
- Clusters tend to have a dominant color: Yellowish if they liked their rated movies a blueish if don't.
- Horizontal lines with the same color correspond to users with low variety in their ratings, they tend to like or dislike most of the movies.

Step 13: Prediction

Choosing a cluster to analyze it and try to make a prediction with it.

Cluster 11

Displaying the data in the form of ratings

Now we will take one of the blank cells, which are movies that haven't been rated by the users, and we will try to predict whether he/she would have liked it or not.

Users are grouped in a cluster with other users that presumably have similar taste to theirs, so it is reasonable to think that he/she would have rated a blank movie with the average of the rest of the users of its cluster.

#Movie Name = "Taxi Driver"

```
In [57]: cluster.fillna('').head()
```

Out[57]:

	Forrest Gump (1994)	Quiz Show (1994)	On Golden Pond (1981)	Get Shorty (1995)	Dead Poets Society (1989)	Wings of Desire (Himmel √¼ber Berlin, Der) (1987)	Blues Brothers, The (1980)	Glengarry Glen Ross (1992)	Monty Python's Life of Brian (1979)	Nightmare Before Christmas, The (1993)	...	Patton (1970)	Snow White and the Seven Dwarfs (1937)	Real Genius (1985)	Crow, The (1994)	Notorious (1946)	Flirting With Disaster (1996)	Ro
0	5.0	3.0	3.0	3.0	3.0	5.0	5.0	5.0	3.0	4.0	...	3	4	4	3	4	3	
1	4.0	3.0	4.0	3.0	4.0	5.0	4.0	4.0	4.0	3.0	...	5	4	4	3			
2	4.0	4.0	5.0	5.0	4.0	5.0	4.0	4.0	4.0	2.0	...					5	1	

3 rows × 300 columns

```
In [39]: #Name of the movie.
movie_name = "Taxi Driver (1976)"

cluster[movie_name].mean()
```

Out[39]: 4.4

```
In [38]: # The average rating of 50 movies as rated by the users in the cluster
cluster.mean().head(50)
```

Out[38]:

```
Dances with Wolves (1990)                        3.927419
Batman (1989)                                    3.300000
True Lies (1994)                                 3.629310
Apollo 13 (1995)                                 3.991379
Pulp Fiction (1994)                              3.964912
Fugitive, The (1993)                             4.072727
Aladdin (1992)                                   3.679245
Beauty and the Beast (1991)                      3.576923
Forrest Gump (1994)                              4.000000
Jurassic Park (1993)                             3.843137
Ace Ventura: Pet Detective (1994)                3.040816
```

Predicted rating for the movie in cluster 11 is 4.4

Step 14: Recommendations

Using the logic of the previous step, Calculating the average score in the cluster for every movie, we will have an understanding for how the cluster feels about each movie in the dataset.

```
# The average rating of 20 movies as rated by the users in the
cluster
cluster.mean().head(20)
```

```
Matrix, The (1999)                                               4.244681
Lord of the Rings: The Two Towers, The (2002)                    4.011111
Dark Knight, The (2008)                                          4.267442
Lord of the Rings: The Return of the King, The (2003)            4.190476
Lord of the Rings: The Fellowship of the Ring, The (2001)        4.047619
Iron Man (2008)                                                  4.105263
Inception (2010)                                                 4.118421
Shrek (2001)                                                     3.723684
Forrest Gump (1994)                                              4.216216
Pirates of the Caribbean: The Curse of the Black Pearl (2003)    3.797297
Batman Begins (2005)                                             4.083333
Incredibles, The (2004)                                          3.657143
Toy Story (1995)                                                 3.720588
Star Wars: Episode IV - A New Hope (1977)                        4.000000
WALL·E (2008)                                                    3.848485
Beautiful Mind, A (2001)                                         3.954545
Monsters, Inc. (2001)                                            3.906250
Fight Club (1999)                                                4.296875
Avatar (2009)                                                    3.790323
Finding Nemo (2003)                                              3.661290
dtype: float64
```

Can recommend movies to users who has not rated yet. In this below example, **Recommending movies to user 19.**

```
user_id = 9

# Get all this user's ratings
user_2_ratings  = cluster.loc[user_id, :]

# Which movies did they not rate?
user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]

# The ratings of these movies the user did not rate
avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inner').loc[:,0]

# Sorting by rating so the highest rated movies are presented first
avg_ratings.sort_values(ascending=False)[:50]
```

Out[31]:

Homeward Bound: The Incredible Journey (1993)	5.000000
Wallace & Gromit: A Close Shave (1995)	5.000000
20,000 Leagues Under the Sea (1954)	5.000000
Contact (1997)	5.000000
Home for the Holidays (1995)	4.500000
From Dusk Till Dawn (1996)	4.500000
Kids in the Hall: Brain Candy (1996)	4.500000
Emma (1996)	4.500000

INTERPRETATION

Above are some of the movies recommended for the user based on the similarity of users i.e. clustered together based on average and sparse rating given by them.

5.1.3 Recommendation System

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

I) Item based Recommender System

OBJECTIVE: To predict movies for different users based on history of movies they watched.

Step 1: Loading Data in R

Dataset stored in CSV format has been loaded in R using read.csv function

```
#-----------------------------------------------------------------------------------------------
# Recommender system
#-----------------------------------------------------------------------------------------------

#Get working Directory
getwd()

#Set working directory
setwd('D:/Fall-19/Data Mining/Project/Final_Presentation/')

# Loading data from CSV file
Movie_ratings=read.csv('ratings_small.csv',header=T)
Movie_data=read.csv('movie.csv',header=T)
head(Movie_data)

#install.packages("recommenderlab", dependencies = TRUE,)
library(recommenderlab)

movierate = Movie_ratings
movierate = as(Movie_ratings, "realRatingMatrix")
```

Step 2: Data split

```
set.seed(101)
trainset=sample(x = c(TRUE, FALSE), size = nrow(ratings_movie1), replace = TRUE,
                prob = c(0.8, 0.2))

MRating_train = ratings_movie1[trainset, ]
MRating_test = ratings_movie1[!trainset, ]

nrow(MRating_train)
nrow(MRating_test)

num_rec = 5  # Lets recommend top 5  movies to each of users
```

Step 3: Build IBCF Models

```
#-------------------------------------------------------------------------------------------------
# Item Based Collaborative Filtering Method
#-------------------------------------------------------------------------------------------------
# Find top 5 recommended movies with Item based collab filter
Rec_IB_Model1 = Recommender(data = MRating_train, method = "IBCF", parameter = list(k = 25,  method = "Cosine"))
Rec_IB_Model1

Model1_IB_Predict = predict(object = Rec_IB_Model1, newdata = MRating_test, n = num_rec,type="topNList")
Model1_IB_Predict

class(Model1_IB_Predict)
```

```
--------------------------------------
> # Item Based Collaborative Filtering Method
> #-------------------------------------------------------------------------
--------------------------------------
> # Find top 5 recommended movies with Item based collab filter
> Rec_IB_Model1 = Recommender(data = MRating_train, method = "IBCF", par
ameter = list(k = 25,  method = "Cosine"))
> Rec_IB_Model1
Recommender of type 'IBCF' for 'realRatingMatrix'
learned using 214 users.
>
> Model1_IB_Predict = predict(object = Rec_IB_Model1, newdata = MRating_
test, n = num_rec,type="topNList")
> Model1_IB_Predict
Recommendations as 'topNList' with n = 5 for 62 users.
>
```

Step 4: Creating User and movie recommendation matrix

```
#-------------------------------------------------------------------------------------------------
# Creating matrix of user vs their top recommended movies
#-------------------------------------------------------------------------------------------------
M1_Total_User=length(Model1_IB_Predict@items)
M1_Toprec_allUser = matrix("null",nrow=M1_Total_User,ncol=num_rec)
dimnames(M1_Toprec_allUser)=list(rownames(M1_Toprec_allUser, do.NULL = FALSE, prefix = "User"),colnames(M1_Top
for (userId in 1:M1_Total_User){
  For_User = Model1_IB_Predict@items[[userId]]
  Movie_Id_Foruser <- Model1_IB_Predict@itemLabels[For_User]
  Movies_Reccom_Foruser <- Movie_Id_Foruser
  for (index in 1:num_rec){
    Movies_Reccom_Foruser[index] <- as.character(subset(Movie_data,Movie_data$movieId == Movie_Id_Foruser[inde
    M1_Toprec_allUser[userId,index]=Movies_Reccom_Foruser[index]
  }
#  IBCF_Movies[=(cbind(userId,M1_Toprec_allUser[userId,]))
}

head(M1_Toprec_allUser)
View(M1_Toprec_allUser,title = "Recommended movies using IBCF")
#-------------------------------------------------------------------
```

Coll_RecSys_MovieLens.R | ContentRecSys_MovieLens.R | SVM_Classification.R | Traditional_MF.R | IBCF_RecSysy.R | Recommended movies using IBCF | UBCF_RecSys.R

Filter

	Movie1	Movie2	Movie3	Movie4	Movie5
User1	Clear and Present Danger (1994)	Pretty Woman (1990)	Clueless (1995)	Bug's Life, A (1998)	Mrs. Doubtfire (1993)
User2	Spider-Man (2002)	Fifth Element, The (1997)	Outbreak (1995)	Stargate (1994)	Net, The (1995)
User3	Ghostbusters (a.k.a. Ghost Busters) (...	Waterworld (1995)	Mission: Impossible (1996)	Titanic (1997)	Clear and Present Danger (1994)
User4	Dead Man Walking (1995)	American History X (1998)	Matrix, The (1999)	American Pie (1999)	Jurassic Park (1993)
User5	Blade Runner (1982)	Twister (1996)	2001: A Space Odyssey (1968)	Pirates of the Caribbean: The Curse of the Black Pearl (2003)	Clear and Present Danger (1994)
User6	Jerry Maguire (1996)	Mask, The (1994)	Rock, The (1996)	True Lies (1994)	Back to the Future (1985)
User7	Cliffhanger (1993)	Spider-Man (2002)	Batman Forever (1995)	Fifth Element, The (1997)	GoldenEye (1995)
User8	Jumanji (1995)	Batman Forever (1995)	Net, The (1995)	Mask, The (1994)	True Lies (1994)
User9	Aladdin (1992)	Spider-Man (2002)	Pirates of the Caribbean: The Curse of...	Jumanji (1995)	Sleepless in Seattle (1993)
User10	Rock, The (1996)	Net, The (1995)	Leaving Las Vegas (1995)	Crouching Tiger, Hidden Dragon (Wo hu cang long) (2000)	Beautiful Mind, A (2001)
User11	Clueless (1995)	Apocalypse Now (1979)	Home Alone (1990)	Ghost (1990)	Austin Powers: The Spy Who Shagged Me (1999)
User12	Spider-Man (2002)	2001: A Space Odyssey (1968)	Bug's Life, A (1998)	Heat (1995)	Leaving Las Vegas (1995)
User13	While You Were Sleeping (1995)	Beautiful Mind, A (2001)	Sleepless in Seattle (1993)	Jumanji (1995)	Toy Story (1995)
User14	Dead Man Walking (1995)	Babe (1995)	Lion King, The (1994)	Aladdin (1992)	Bug's Life, A (1998)
User15	Sleepless in Seattle (1993)	Apollo 13 (1995)	Shakespeare in Love (1998)	Pretty Woman (1990)	While You Were Sleeping (1995)
User16	Clueless (1995)	Raiders of the Lost Ark (Indiana Jones an...	Beetlejuice (1988)	Kill Bill: Vol. 1 (2003)	Beautiful Mind, A (2001)

Step 5: Building model to find Accuracy

```
## predict ratings for new users
Predict_Rating = predict(Rec_IB_Model1, MRating_test, type="ratings")
Predict_Rating@data
as(Predict_Rating, "matrix")[,1:10]
#-------------------------------------------------------------------------------------------------------------------
#ACCURACY
#-------------------------------------------------------------------------------------------------------------------
#Building Models
model_IB = Recommender(MRating_train, method = "IBCF",param=list(normalize = "center", method="Cosine", k=30))

#Making predictions
prediction = predict(model_IB, MRating_test[1:5], type="ratings")
as(prediction, "matrix")[,1:8]
```

Step 6: Finding Accuracy using RMSE

```
#Estimating RMSE
set.seed(101)

eval_scheme_IB = evaluationScheme(ratings_movie1, method="split", train=0.8, given=10,goodRating=5)

eval_model_IB = Recommender(getData(eval_scheme_IB, "train"), method = "IBCF",
                            param=list(normalize = "center", method="Cosine", k=30))

Prediction_IB = predict(eval_model_IB, getData(eval_scheme_IB, "known"), type="ratings")

#RMSE for an entire model
IBCF_Fullaccuracy = calcPredictionAccuracy(Prediction_IB, getData(eval_scheme_IB, "unknown"))
IBCF_AccuracyByUser = calcPredictionAccuracy(Prediction_IB, getData(eval_scheme_IB, "unknown"),byUser = TRUE)
#-------- IBCF ------------
print(IBCF_Fullaccuracy)
print(head(IBCF_AccuracyByUser))
```

	IBCF
RMSE	1.2061923
MSE	1.4548999
MAE	0.9028126

	RMSE	MSE	MAE
15	1.6549764	2.7389468	1.3562472
20	1.8279813	3.3415155	1.3227687
21	0.9411506	0.8857644	0.6802556
34	0.8505026	0.7233547	0.6459166
42	0.7659740	0.5867161	0.5952808
59	1.9388585	3.7591724	1.5762285

Step 7: Finding Precision and Recall

```
#-------------------------------------------------------------------------------------------------
#Precision and Recall
#-------------------------------------------------------------------------------------------------

#install.packages("precrec")
library(precrec)

Evaluate_result_IBCF = evaluate(eval_scheme_IB,method="IBCF",n=c(1,3,5,10,15,20))
Evaluate_IBCF = getConfusionMatrix(Evaluate_result_IBCF)[[1]]
print(Evaluate_IBCF)
kable(print(Evaluate_IBCF),caption = "User Rating Prediction accuracy(Sample) - IBCF")
```

	TP	FP	FN	TN	precision	recall	TPR	FPR
1	0.0178571	0.9821429	9.982143	128.0179	0.0178571	0.0008117	0.0008117	0.0076341
3	0.1428571	2.8571429	9.857143	126.1429	0.0476190	0.0101765	0.0101765	0.0221677
5	0.2321429	4.7678571	9.767857	124.2321	0.0464286	0.0338208	0.0338208	0.0370273
10	0.8035714	9.1964286	9.196429	119.8036	0.0803571	0.1066760	0.1066760	0.0713893
15	1.0714286	13.9285714	8.928571	115.0714	0.0714286	0.1287459	0.1287459	0.1081088
20	1.5000000	18.5000000	8.500000	110.5000	0.0750000	0.1664199	0.1664199	0.1434795

INTERPRETATION

```
> # USER_ID = 10
> Movies_Reccom_Foruser
[1] "Rock, The (1996)"
[2] "Net, The (1995)"
[3] "Leaving Las Vegas (1995)"
[4] "Crouching Tiger, Hidden Dragon (Wo hu cang long) (2000)"
[5] "Beautiful Mind, A (2001)"
```

Above are some of the movies recommended for the user based on their items using topN recommendation with an overall RMSE of 1.20 for user 10.

II) User based Recommender System

OBJECTIVE: To predict movies for different users based on history of movies they watched and comparing their genre preference with another user.

Step 1: Loading Data in R

Dataset stored in CSV format has been loaded in R using read.csv function

```
#-------------------------------------------------------------------------------------------------
# Recommender system
#-------------------------------------------------------------------------------------------------

#Get working Directory
getwd()

#Set working directory
setwd('D:/Fall-19/Data Mining/Project/Final_Presentation/')

# Loading data from CSV file
Movie_ratings=read.csv('ratings_small.csv',header=T)
Movie_data=read.csv('movie.csv',header=T)
head(Movie_data)

#install.packages("recommenderlab", dependencies = TRUE,)
library(recommenderlab)

movierate = Movie_ratings
movierate = as(Movie_ratings, "realRatingMatrix")
```

Step 2: Data split

```
set.seed(101)
trainset=sample(x = c(TRUE, FALSE), size = nrow(ratings_movie1), replace = TRUE,
                prob = c(0.8, 0.2))

MRating_train = ratings_movie1[trainset, ]
MRating_test = ratings_movie1[!trainset, ]

nrow(MRating_train)
nrow(MRating_test)

num_rec = 5  # Lets recommend top 5  movies to each of users
```

Step 3: Build UBCF Models

```
#-------------------------------------------------------------------------------------------------
# User Based Collaborative Filtering Method
#-------------------------------------------------------------------------------------------------
# Find top 5 recommended movies with Item based collab filter
#200 users

Rec_Ub_Model1 = Recommender(data = MRating_train, method = "UBCF", parameter = list(k = 25, method = "Cosine"))
Rec_Ub_Model1

#Recommender of type 'UBCF' for 'realRatingMatrix' learned using 214 users.
Model1_Ub_Predict = predict(object = Rec_Ub_Model1, newdata = MRating_test, n = num_rec,type="topNList")
Model1_Ub_Predict

#
```

```
> # User Based Collaborative Filtering Method
> #------------------------------------------------------------------------------------------
------------------
> # Find top 5 recommended movies with Item based collab filter
> #200 users
>
> Rec_Ub_Model1 = Recommender(data = MRating_train, method = "UBCF", parameter = list(k =
 25, method = "Cosine"))
Warning: Unknown parameters: k
Available parameter (with default values):
method   =  cosine
nn       =  25
sample   =  FALSE
normalize        =  center
verbose  =  FALSE
> Rec_Ub_Model1
Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 214 users.
>
> #Recommender of type 'UBCF' for 'realRatingMatrix' learned using 214 users.

> Model1_Ub_Predict = predict(object = Rec_Ub_Model1, newdata = MRating_test, n = num_re
c,type="topNList")
> Model1_Ub_Predict
Recommendations as 'topNList' with n = 5 for 62 users.
> Model1_Ub_Predict
```

Step 4: Creating User and movie recommendation matrix

```
#-------------------------------------------------------------------------------------
# Creating matrix of user vs their top recommended movies
#-------------------------------------------------------------------------------------
M2_Total_User=length(Model1_Ub_Predict@items)
M2_Toprec_allUser = matrix("null",nrow=M2_Total_User,ncol=num_rec)
dimnames(M2_Toprec_allUser)=list(rownames(M2_Toprec_allUser, do.NULL = FALSE, prefix = "User"),colnames(M2_Toprec_allUs
for (userId in 1:M2_Total_User){
  For_User = Model1_Ub_Predict@items[[userId]]
  Movie_Id_Foruser <- Model1_Ub_Predict@itemLabels[For_User]
  Movies_Reccom_Foruser <- Movie_Id_Foruser
  for (index in 1:num_rec){
    Movies_Reccom_Foruser[index] <- as.character(subset(Movie_data,Movie_data$movieId == Movie_Id_Foruser[index])$title
    M2_Toprec_allUser[userId,index]=Movies_Reccom_Foruser[index]
  }
}
head(M2_Toprec_allUser)
View(M2_Toprec_allUser,title = "Recommended movies using UBCF")

#
```

Coll_RecSys_MovieLens.R | ContentRecSys_MovieLens.R | SVM_Classification.R | Traditional_MF.R | IBCF_RecSysy.R | UBCF_RecSys.R | Recommended movies using UBCF

Filter

	Movie1	Movie2	Movie3	Movie4	Movie5
User1	Usual Suspects, The (1995)	Shawshank Redemption, The (1994)	Trainspotting (1996)	Fargo (1996)	Braveheart (1995)
User2	Clueless (1995)	LÃ©on: The Professional (a.k.a. The Professional) (LÃ©on) (...	Dark Knight, The (2008)	Trainspotting (1996)	Amelie (Fabuleux destin d'AmÃ©lie F
User3	Usual Suspects, The (1995)	Star Wars: Episode V - The Empire Strikes Back (1980)	Fargo (1996)	Casablanca (1942)	Star Wars: Episode IV - A New Hope
User4	Pulp Fiction (1994)	Trainspotting (1996)	Silence of the Lambs, The (1991)	Usual Suspects, The (1995)	American Beauty (1999)
User5	Pulp Fiction (1994)	Usual Suspects, The (1995)	Star Wars: Episode V - The Empire Strikes Back (1980)	Schindler's List (1993)	Back to the Future (1985)
User6	Silence of the Lambs, The (1991)	Shawshank Redemption, The (1994)	Schindler's List (1993)	Apollo 13 (1995)	Usual Suspects, The (1995)
User7	Usual Suspects, The (1995)	American Beauty (1999)	Amelie (Fabuleux destin d'AmÃ©lie Poulain, Le) (2001)	Star Wars: Episode VI - Return of th...	Godfather, The (1972)
User8	Monty Python and the Holy Grail (1...	Lord of the Rings: The Fellowship of the Ring, The (2001)	Usual Suspects, The (1995)	Raiders of the Lost Ark (Indiana Jon...	Schindler's List (1993)
User9	Fight Club (1999)	Lord of the Rings: The Fellowship of the Ring, The (2001)	Lord of the Rings: The Return of the King, The (2003)	Memento (2000)	Toy Story (1995)
User10	Usual Suspects, The (1995)	Dr. Strangelove or: How I Learned to Stop Worrying and Lov...	Schindler's List (1993)	Apollo 13 (1995)	Taxi Driver (1976)
User11	Schindler's List (1993)	Shawshank Redemption, The (1994)	Forrest Gump (1994)	Godfather, The (1972)	Apollo 13 (1995)
User12	Schindler's List (1993)	Usual Suspects, The (1995)	Godfather, The (1972)	Pulp Fiction (1994)	Princess Bride, The (1987)
User13	Shawshank Redemption, The (1994)	Schindler's List (1993)	Pulp Fiction (1994)	Usual Suspects, The (1995)	Forrest Gump (1994)
User14	Godfather, The (1972)	Godfather: Part II, The (1974)	Goodfellas (1990)	Casablanca (1942)	Schindler's List (1993)
User15	Shawshank Redemption, The (1994)	Godfather, The (1972)	Silence of the Lambs, The (1991)	Fargo (1996)	Godfather: Part II, The (1974)
User16	Terminator 2: Judgment Day (1991)	Lord of the Rings: The Return of the King, The (2003)	Lord of the Rings: The Fellowship of the Ring, The (2001)	Lord of the Rings: The Two Towers, ...	Monty Python and the Holy Grail (19
User17	Usual Suspects, The (1995)	Fargo (1996)	Seven (a.k.a. Se7en) (1995)	Apocalypse Now (1979)	Goodfellas (1990)

Step 5: Building model and finding Accuracy

```
#---------------------------------------------------------------------------------------------------------------
#ACCURACY
#---------------------------------------------------------------------------------------------------------------

#Building Models
model = Recommender(MRating_train, method = "UBCF", param=list(normalize = "center", method="Cosine"))

#Making predictions
#Define a matrix with the recommendations for each user. I visualize the recommendations for the first four users:
prediction = predict(model, MRating_test[1:5], type="ratings")
as(prediction, "matrix")[,1:8]

#Estimating RMSE
set.seed(1)
eval_scheme_UB = evaluationScheme(ratings_movie1, method="split",train=0.8, given=10,goodRating=5)
eval_model_UB = Recommender(getData(eval_scheme_UB, "train"), method = "UBCF",param=list(normalize = "center", method=
Prediction_UB = predict(eval_model_UB, getData(eval_scheme_UB, "known"), type="ratings")

#RMSE for an entire model

UBCF_Fullaccuracy = calcPredictionAccuracy(Prediction_UB, getData(eval_scheme_UB, "unknown"),byUser = FALSE)
UBCF_AccuracyByUser = calcPredictionAccuracy(Prediction_UB, getData(eval_scheme_UB, "unknown"),byUser = TRUE)
```

Step 6: Finding Accuracy using RMSE

	UBCF
RMSE	0.8570230
MSE	0.7344885
MAE	0.6644531

	RMSE	MSE	MAE
2	0.6359623	0.4044480	0.5616437
4	0.6093972	0.3713650	0.3299118
8	0.6663461	0.4440172	0.5619457
21	0.7463398	0.5570231	0.6447048
26	0.9678791	0.9367900	0.7320927
34	0.7810796	0.6100853	0.6226973

Step 7: Finding Precision and Recall

```
#---------------------------------------------------------------------------------------------------------------
#Precision and Recall
#---------------------------------------------------------------------------------------------------------------

install.packages("precrec")
library(precrec)

Evaluate_result_UBCF = evaluate(eval_scheme_UB,method="UBCF",n=c(1,3,5,10,15,20))
Evaluate_UBCF = getConfusionMatrix(Evaluate_result_UBCF)[[1]]
print(Evaluate_UBCF)

kable(print(Evaluate_UBCF),caption = "User Rating Prediction accuracy(Sample) - UBCF")
```

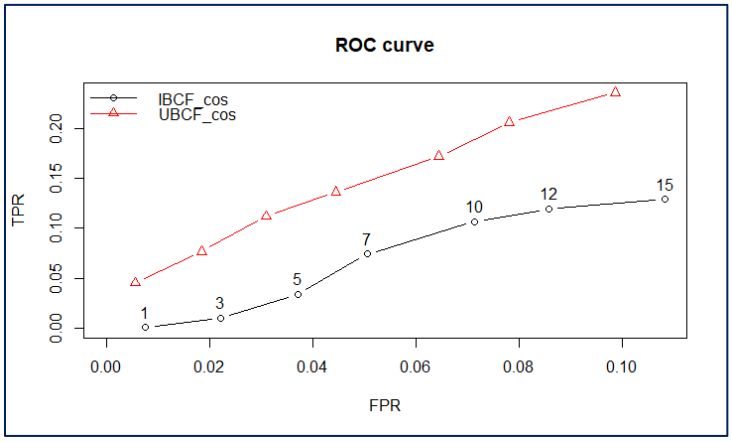
	TP	FP	FN	TN	precision	recall	TPR	FPR
1	0.2142857	0.7857143	9.500000	128.5000	0.2142857	0.0196749	0.0196749	0.0059985
3	0.4821429	2.5178571	9.232143	126.7679	0.1607143	0.0472885	0.0472885	0.0193397
5	0.7857143	4.2142857	8.928571	125.0714	0.1571429	0.0844802	0.0844802	0.0324040
10	1.4642857	8.5357143	8.250000	120.7500	0.1464286	0.1514183	0.1514183	0.0657065
15	2.0000000	13.0000000	7.714286	116.2857	0.1333333	0.1990054	0.1990054	0.1001770
20	2.5357143	17.4642857	7.178571	111.8214	0.1267857	0.2492031	0.2492031	0.1345977

INTERPRETATION

```
> # USER_ID = 10
> Movies_Reccom_Foruser
[1] "Usual Suspects, The (1995)"
[2] "Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)"
[3] "Schindler's List (1993)"
[4] "Apollo 13 (1995)"
[5] "Taxi Driver (1976)"
```

Above are some of the movies recommended for the user based on their behavior and the history using topN recommendation with an overall RMSE of 0.83 for user 10.

To compare both models in collaborative recommender system we plotted ROC curve between User Based Collaborative Filtering (UBCF) and Item Based Collaborative Filtering (IBCF). From the curve we could see that area under curve (AUC) for UBCF is more than IBCF.



III) Recommender System using Matrix Factorization

OBJECTIVE: To predict movies for different users based on user-item interaction matrix.

Step 1: Loading Data in R

Dataset stored in CSV format has been loaded in R using read.csv function

```
#-------------------------------------------------------------------------------
#  MATRIX FACTORIZATION
#-------------------------------------------------------------------------------
#Get working Directory
getwd()

#Set working directory
setwd('D:/Fall-19/Data Mining/Project/Movielens_Project/movielens-20m-dataset/')

# Loading data from CSV file
Movie_ratings=read.csv('ratings_small.csv',header=T)
Movie_data=read.csv('movie.csv',header=T)

head(Movie_data)
#Matrix_Rating = Movie_ratings %>% dcast(formula = movieId ~ userId) %>% as.matrix()
head(Movie_ratings)
#install.packages("recosystem",dependencies = TRUE)
library(recosystem)
```

Step 2: Build Matrix Factorization Models

Model 1:

```
#-------------------------------------------------------------------------------------------------------------------------
# Matrix Factorization
#-------------------------------------------------------------------------------------------------------------------------
MF_recom=Reco()
#Model 1
MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rating)
opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),costp_l1 = 0, costq_l1 = 0,nthread = 1, niter = 10))
MF_recom$train(MF_Trainset, opts = opts)

#Model 2:
```

Model 2:

```
#Model 2
MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rating)
opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(10, 20, 30), lrate = c(0.7, 0.8),costp_l1 = 0, costq_l1 = 0,nthread = 1, niter = 10))
MF_recom$train(MF_Trainset, opts = opts)
```

Model 3:

```
#Model 3
MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rating)
opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(10, 20, 30), lrate = c(0.4, 0.5),costp_l1 = 0, costq_l1 = 0,nthread = 1, niter = 10))
MF_recom$train(MF_Trainset, opts = opts)
```

```
> # Matrix Factorization
> #-------------------------------------------------------------------------------------------------------------------------
> MF_recom=Reco()
> MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rat
ing)
> opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(10, 20, 30), lrate = c(0.4, 0.5),costp_l1 =
 0, costq_l1 = 0,nthread = 1, niter = 10))
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
> MF_recom$train(MF_Trainset, opts = opts)
iter      tr_rmse          obj
   0       1.3536  2.5892e+005
   1       0.9104  1.5748e+005
   2       0.8786  1.5160e+005
   3       0.8614  1.4842e+005
   4       0.8482  1.4611e+005
   5       0.8360  1.4409e+005
   6       0.8251  1.4254e+005
   7       0.8147  1.4112e+005
   8       0.8064  1.4010e+005
   9       0.7983  1.3876e+005
  10       0.7920  1.3803e+005
  11       0.7859  1.3722e+005
  12       0.7809  1.3663e+005
  13       0.7764  1.3611e+005
  14       0.7720  1.3557e+005
  15       0.7686  1.3515e+005
  16       0.7654  1.3476e+005
  17       0.7630  1.3457e+005
```

Model 4:

```
#Model 4
MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rating)
opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(30, 40, 50), lrate = c(0.4, 0.5),costp_l1 = 0, costq_l1 = 0,nthread = 1, niter = 10))
MF_recom$train(MF_Trainset, opts = opts)

#Model 5
```

Model 5:

```
#Model 5
MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rating)
opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(20, 30, 40), lrate = c(0.5, 0.6),costp_l1 = 0, costq_l1 = 0,nthread = 1, niter = 10))
MF_recom$train(MF_Trainset, opts = opts)
```

Model 6:

```
#Model 6
MF_Trainset = data_memory(Movie_ratings$userId, Movie_ratings$movieId, rating = Movie_ratings$rating)
opts = MF_recom$tune(MF_Trainset, opts = list(dim = c(10, 20, 30), lrate = c(0.5, 0.6),costp_l1 = 0, costq_l1 = 0,nthread = 1, niter = 10))
MF_recom$train(MF_Trainset, opts = opts)
```

Step 4: Creating User and movie recommendation matrix

```
#-------------------------------------------------------------------------------------------------------------------------
# Creating matrix of user vs their top recommended movies
#-------------------------------------------------------------------------------------------------------------------------
#install.packages("magrittr")
library(magrittr)
library(data.table)
num_rec=5
#M3_Total_User=length(Model3_MF_Predict@items)
M3_Total_User=62
M3_Toprec_allUser = matrix("null",nrow=M3_Total_User,ncol=num_rec)
dimnames(M3_Toprec_allUser)=list(rownames(M3_Toprec_allUser, do.NULL = FALSE, prefix = "User"),colnames(M3_Toprec_allUser,do.
for (user in 1:M3_Total_User){
  MF_Testset  =  data_memory(rep(c(user),nrow(Movie_data)), Movie_data$movieId, rating = NULL)
  temp=(data.frame(userId = user,movieId = Movie_data$title,rating = MF_recom$predict(MF_Testset, out_memory())) %>%setorder(
#   print(temp)
  for (index in 1:num_rec){
    M3_Toprec_allUser[user,index]=as.character(temp[index,2])
  }
}
head(M3_Toprec_allUser[25,])
View(M3_Toprec_allUser,title = "Recommended movies using MF")
#-------------------------------------------------------------------------------------------------------------------------
```

MF_RecSysy.R | Recommended movies using MF | UBCF_RecSys.R | KNNClassifi-IMDB.R | SVM_Classification.R

Filter

	Movie1	Movie2	Movie3	Movie4	Movie5
User1	Run, Man, Run! (Corri uomo corri) (1968)	Call Me Bwana (1963)	'Twas the Night Before Christmas (1974)	New Wave (Nouvelle vague) (1...	Swarm, The (1978)
User2	Perfect Man, The (2005)	Vesna va veloce (1996)	Cet amour-lÃ (2001)	T.N.T. (1997)	Little Girl Who Lives Down the Lane, The (1976)
User3	Shriek If You Know What I Did Last Friday th...	No Regrets for Our Youth (Waga seish...	Xtro (1983)	Life Happens (2012)	Hotel (2013)
User4	Sachs' Disease (La maladie de Sachs) (1999)	Living Wake, The (2007)	I'm Gonna Explode (a.k.a. I'm Going to ...	Fort McCoy (2014)	This is Our Time (2013)
User5	Fast & Furious 6 (Fast and the Furious 6, The...	Wait Until Dark (1967)	Gozu (GokudÃ´ kyÃ´fu dai-gekijÃ´: Goz...	Iron Giant, The (1999)	Pale Rider (1985)
User6	Walking with Prehistoric Beasts (2001)	Zorro, the Gay Blade (1981)	Hollywood Shuffle (1987)	9 Souls (Nain souruzu) (2003)	Long Time Dead (2002)
User7	Sweepers (1998)	Curse of the Cat People, The (1944)	You Are So Beautiful (Je vous trouve tr...	So I Married an Axe Murderer (...	Tupac: Resurrection (2003)
User8	Care Bears Movie II: A New Generation (1986)	Won Ton Ton: The Dog Who Saved H...	Fubar (2002)	The Wedding Ringer (2015)	Body of War (2007)
User9	When the Game Stands Tall (2014)	Sincerely Yours (1955)	Club Paradise (1986)	Le crocodile du Botswanga (20...	Good Night to Die, A (2003)
User10	Hypnotist, The (HypnotisÃ¶ren) (2012)	I Am Waiting (Ore wa matteru ze) (19...	Maybe, Maybe Not (Bewegte Mann, De...	Great Mouse Detective, The (1...	Saving Face (2004)
User11	Woman in White, The (1948)	Story of the Weeping Camel, The (Ges...	Trouble at Timpetill (Enfants de Timpel...	Unconscious (Inconscientes) (2...	Borrowers, The (2011)
User12	Journey, The (1959)	Major Movie Star (2008)	Forever Hardcore: The Documentary (2...	In This World (2002)	Somewhere in the Night (1946)
User13	Dragon Inn (Sun lung moon hak chan) (1992)	Back to the Garden, Flower Power Co...	Your Friends and Neighbors (1998)	Take a Girl Like You (1970)	Killing, The (1956)
User14	Butley (1974)	Brokeback Mountain (2005)	Balloonatic, The (1923)	Life with Mikey (1993)	My Little Pony: Equestria Girls (2013)
User15	Wives and Lovers (1963)	Performance (1970)	Electra Glide in Blue (1973)	Smell of Camphor, Fragrance o...	Another Earth (2011)
User16	New Best Friend (2002)	French Kiss (1995)	Magnificent Yankee, The (1950)	First Day of the Rest of Your Lif...	Get Smart (2008)
User17	Descendants, The (2011)	Union, The (2011)	Yu-Gi-Oh! (1999)	Maria (Mariya) (1989)	The Trap (1946)

Step 5: Finding Accuracy using RMSE for Model3

```
#-------------------------------------------------------------------------------------------------------------------------
#ACCURACY
#-------------------------------------------------------------------------------------------------------------------------

eval_model_MF <- predict(Rec_MF_Model3, getData(eval_scheme_MF, "known"), type = "ratings")
head(eval_model_MF@data)
getRatingMatrix(eval_model_MF)[1:6,1:6]

MF_Fullaccuracy = calcPredictionAccuracy(eval_model_MF, getData(eval_scheme_MF, "unknown"))
MF_accuracyByuser = calcPredictionAccuracy(eval_model_MF, getData(eval_scheme_MF, "unknown"),byUser = TRUE)
#-------------------------------------------------------------------------
print(MF_Fullaccuracy)
print(head(MF_accuracyByuser))
#-------------------------------------------------------------------------------------------------------------------------
```

	MF
RMSE	0.9684374
MSE	0.9378709
MAE	0.7435092

	RMSE	MSE	MAE
3	0.8820188	0.7779571	0.6961515
5	0.5746201	0.3301883	0.4438642
9	0.9050015	0.8190277	0.7166184
22	1.0051555	1.0103377	0.8811526
23	0.8775628	0.7701165	0.6910134
33	1.0852832	1.1778395	0.9370434

Step 7: Finding Precision and Recall

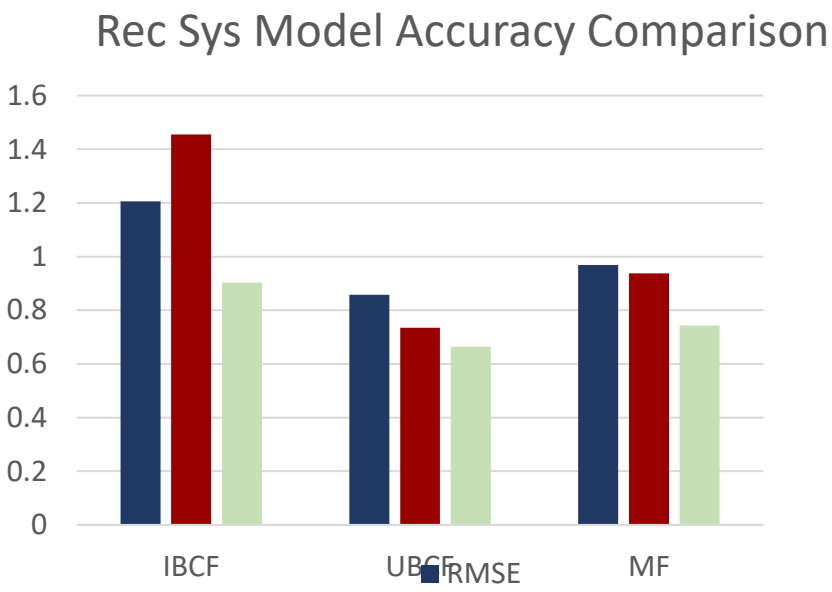
```
#-------------------------------------------------------------------------------
#Precision and Recall
#-------------------------------------------------------------------------------

Evaluate_result_MF = evaluate(eval_scheme_MF,method="SVD",n=c(1,3,5,10,15,20))
Evaluate_MF = getConfusionMatrix(Evaluate_result_MF)[[1]]
print(Evaluate_MF)
```

	TP	FP	FN	TN	precision	recall	TPR	FPR
1	0.1185185	0.8814815	22.58519	9032.415	0.11851852	0.01732597	0.01732597	9.756991e-05
3	0.3407407	2.6592593	22.36296	9030.637	0.11358025	0.03617808	0.03617808	2.943286e-04
5	0.5333333	4.4666667	22.17037	9028.830	0.10666667	0.04745933	0.04745933	4.943762e-04
10	1.0592593	8.9407407	21.64444	9024.356	0.10592593	0.07982749	0.07982749	9.895567e-04
15	1.4074074	13.5925926	21.29630	9019.704	0.09382716	0.10071975	0.10071975	1.504444e-03
20	1.7111111	18.2888889	20.99259	9015.007	0.08555556	0.12006787	0.12006787	2.024225e-03

INTERPRETATION

Models	RMSE	MSE	MAE
Model 1	0.9825301	0.9653653	0.7606691
Model 2	1.0241410	1.0488648	0.8061399
Model 3	0.9684374	0.9378709	0.7435092
Model 4	1.0101647	1.0204328	0.7782772
Model 5	1.0205222	1.0414655	0.7962625
Model 6	0.9756178	0.9518301	0.7536567

We created various model by changing dimension and latency rate and upon looking into the table, among the models built Model3 has the least root mean square error.



On comparing three collaborative recommender system models we could see that UBCF performs good for given data set and Matrix factorization accuracy is slightly close with UBCF.

IV) Content based Recommender System

OBJECTIVE: To recommend movies based on a comparison between the content of the movies and a user profile.

Step 1: Loading Data in R

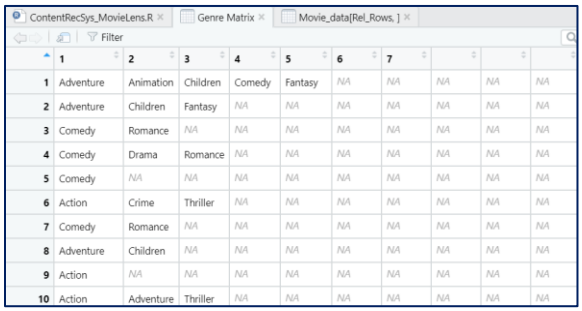
Dataset stored in CSV format has been loaded in R using read.csv function

```
#-------------------------------------------------------------------------------
# Content Based Filtering Method
#-------------------------------------------------------------------------------
#Getting present working Directory
getwd()
#Modifying working directory

setwd('D:/Fall-19/Data Mining/Project/Movielens_Project/movielens-20m-dataset/')

# Loading data from CSV file
#Movie_ratings=read.csv('rating.csv',header=T)
Movie_ratings=read.csv('ratings_small.csv',header=T)
Movie_data=read.csv('movie.csv',header=T)
Movie_Genres = as.data.frame(Movie_data$genres, stringsAsFactors=FALSE)
```

Step 2: Data Preprocessing – Making Genre Matrix

Extracting each genre from the pool of genre mentioned for each movie.

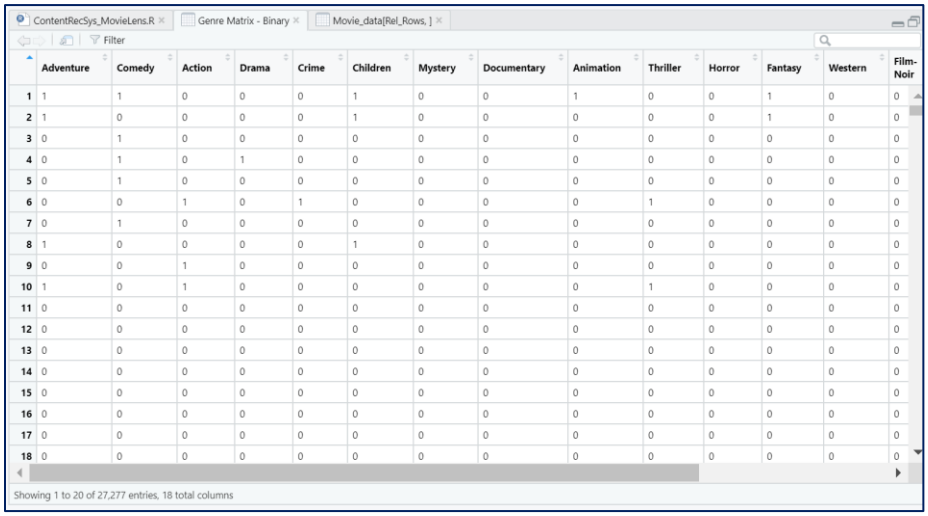
```
# Dealing with Genre
#install.packages("data.table")
library(data.table)
Genre = as.data.frame(tstrsplit(Movie_Genres[,1], '[|]', type.convert=TRUE), stringsAsFactors=FALSE)
head(Genre)
colnames(Genre) = c(1:7)
View(Genre, title = "Genre Matrix")
Total_Genre=print(unlist(Genre))
Genre_List = unique(Total_Genre)
print(Genre_List)

# Removing junk values
Genre_List = Genre_List[-19]
Genre_List = Genre_List[-19]
Genre_List = Genre_List[-19]
dim(Genre)
Genre_Matrix_Bin = matrix(0,27278,18)
Genre_Matrix_Bin[1,] = Genre_List
colnames(Genre_Matrix_Bin) = Genre_List
head(Genre_Matrix_Bin[1,])
dim(Genre_Matrix_Bin)
```



	1	2	3	4	5	6	7
1	Adventure	Animation	Children	Comedy	Fantasy	NA	NA
2	Adventure	Children	Fantasy	NA	NA	NA	NA
3	Comedy	Romance	NA	NA	NA	NA	NA
4	Comedy	Drama	Romance	NA	NA	NA	NA
5	Comedy	NA	NA	NA	NA	NA	NA
6	Action	Crime	Thriller	NA	NA	NA	NA
7	Comedy	Romance	NA	NA	NA	NA	NA
8	Adventure	Children	NA	NA	NA	NA	NA
9	Action	NA	NA	NA	NA	NA	NA
10	Action	Adventure	Thriller	NA	NA	NA	NA

Step 3: Converting Genre matrix into binary values

```
# Convert Genre matrix into dataframe
Genre_Matrix_DF <- as.data.frame(Genre_Matrix_Bin[-1,], stringsAsFactors=FALSE) #remove first row,

# Convert from characters to integers in Genre matrix
for (c in 1:ncol(Genre_Matrix_DF)) {
  Genre_Matrix_DF[,c] <- as.integer(Genre_Matrix_DF[,c])
}
```



	Adventure	Comedy	Action	Drama	Crime	Children	Mystery	Documentary	Animation	Thriller	Horror	Fantasy	Western	Film-Noir
1	1	1	0	0	0	1	0	0	1	0	0	1	0	0
2	1	0	0	0	0	1	0	0	0	0	0	1	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	1	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	0	0	0	0	0
10	1	0	1	0	0	0	0	0	0	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Showing 1 to 20 of 27,277 entries, 18 total columns

Step 4: Normalizing rating

```
# Dealing with rating
Normalize_Ratings = Movie_ratings

# Assinging rating >3 as 1 and <=3 as -1
for (i in 1:nrow(Normalize_Ratings)){
  if (Normalize_Ratings[i,3] > 3){
    Normalize_Ratings[i,3] = 1
  }
  else{
    Normalize_Ratings[i,3] = -1
  }
}
```

Step 5: Creating User profile

```
#Calculate dot product for User Profiles

User_Prof = matrix(0,18,9066) # 706 ???
for (c in 1:ncol(Binary_Ratings)){
  for (i in 1:ncol(Genre_Matrix)){
    User_Prof[i,c] = sum((Genre_Matrix[,i]) * (Binary_Ratings[,c]))
  }
}

dim(User_Prof)
head(User_Prof)

#Convert to Binary scale
for (i in 1:nrow(User_Prof)){
  if (User_Prof[i] < 0){
    User_Prof[i] = 0
  }
  else {
    User_Prof[i] = 1
  }
}
```

ContentRecSys_MovieLens.R | User Profile | Movie_data[Rel_Rows,]

Filter | Cols: « < 1 - 50 > »

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17
1	0	0	0	0	0	0	-1	0	1	0	0	0	1	0	-1	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	-1	0	1	0	0	0	1	0	-1	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Step 5: Build Content Based rec-sys Models – Using Euclidean distance

```
# Framing similarity matrix for selected user wrt his genre pref
sim_mat = rbind.data.frame(For_User, Genre_Matrix)

#convert data to type integer
sim_mat = data.frame(lapply(sim_mat,function(x){as.integer(x)}))

gc()
# Distance measure jaccard, cosine not working
sim_results = dist(sim_mat, method = "euclidean")

sim_results <- as.data.frame(as.matrix(sim_results[1:8552])) ## 8552? ??
Rel_Rows <- which(sim_results == min(sim_results))
head(sim_results)
#Recommended movies
View(Movie_data[Rel_Rows,], title = "Content Based Movie Rec - User1")
head(Movie_data[Rel_Rows,])
```

Step 6: Creating personalized movie recommendation for User1

ContentRecSys_MovieLens.R | Content Based Movie Rec - User1 | Movie_data[Rel_Rows,]

Filter

	movieId	title	genres
10	10	GoldenEye (1995)	Action|Adventure|Thriller
11	11	American President, The (1995)	Comedy|Drama|Romance
12	12	Dracula: Dead and Loving It (1995)	Comedy|Horror
13	13	Balto (1995)	Adventure|Animation|Children
14	14	Nixon (1995)	Drama
15	15	Cutthroat Island (1995)	Action|Adventure|Romance
16	16	Casino (1995)	Crime|Drama
17	17	Sense and Sensibility (1995)	Drama|Romance
18	18	Four Rooms (1995)	Comedy
19	19	Ace Ventura: When Nature Calls (1995)	Comedy
20	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller
21	21	Get Shorty (1995)	Comedy|Crime|Thriller
22	22	Copycat (1995)	Crime|Drama|Horror|Mystery|Thriller
23	23	Assassins (1995)	Action|Crime|Thriller
24	24	Powder (1995)	Drama|Sci-Fi
25	25	Leaving Las Vegas (1995)	Drama|Romance
26	26	Othello (1995)	Drama
27	27	Now and Then (1995)	Children|Drama

INTERPRETATION

Above are some of the top recommended movies based on the content and the behavior of the user.

6. Conclusion

Thus, these models will help users and website to recommend the movies based on customer preference in past and another users' preference in genre. Hence, they can use this information to achieve maximum profit, promote their products across different kind of customers and can make personalized recommendation for customer.

7. Limitation

As most of the features mentioned in IMDB data set are related to popularity, we need to exclude those features to avoid cold start problem. Hence,

- To try various classification techniques
- To predict any features other than Rating.

8. Future Work

- To build hybrid rec-models using traditional approaches.
- To frame context aware rec-models and compare its performance with others.
- To explore, build Switching rec-models which switch between different algorithms and use the algorithm to have the best result.