Movies Recommendation

Ana Sofia Medeiros Oliveira (201503082)
Fábio Henrique da Silva Pereira (201506051)

Introduction

A recommendation system is a system that tries to predict what a user will like or want to buy, for example. Recommendation systems are important as they facilitate the consumers' lives, bringing them the products they want to, well, consume (sometimes even before the user knowing about the product's existence). Recommendation systems have seen their economic worth growing, with some big tech companies registering profit in the order of thousands of millions of dollars due to these systems. 1 2

In this work, we compare the performance of different recommendation strategies for movies' reviews. To achieve that, we'll be using datasets obtained from RottenTomatoes, a online aggregator of movie and TV show reviews from critics.

In this report, we first describe and preprocess the available data. We also provide some data visualization on some of the most important attributes of the data. Next, we describe the modelling approaches that will be used to build the recommendation systems. We then divide the project in two sections – recommendation systems using binary ratings and using real ratings. Subsequently, we implement context into these systems, where we consider attributes like the date of the reviews and the genre of the movie. Finally, we give conclusions about the achieved results, shortcomings and possible improvements to this work.

Data

Two different datasets were used in this work: reviews and movies. As the movies dataset played an auxiliary role in this project, we will only provide insights on the reviews dataset. In respect to the movies dataset, the preprocessing was done when needed.

```
library(dplyr)
library(ggplot2)
library(recommenderlab)
library(tibble)
library(purrr)
library(tidyr)
library(lubridate)

movies <- read.csv("movie_info.tsv", sep="\t")
reviews <- read.csv("reviews.tsv", sep="\t")
```

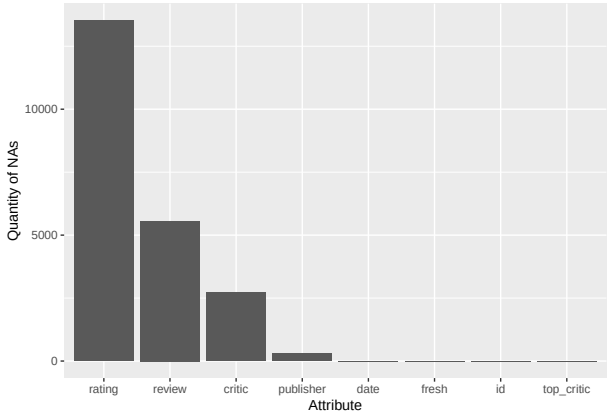
Understanding and Preprocessing

The main dataset is composed by more than 50 thousand instances and 8 different attributes:

- id – identification number of the movie in question
- review – the user's review of the movie
- rating – rating that the user gave to the movie
- fresh – fresh status of the movie (fresh or rotten)
- critic – name of the user

- top_critic – if the user is considered to be a top critic or not
- publisher – site where the review was published
- date – when the review was published

Among these attributes, there were some which had a significant number of missing values. Considering the empty string to be a missing value as well, it can be seen in the next figure the quantity of missing values in each attribute.



As we're building a recommendation system, it doesn't make much sense to have users with no name, so we decided to eliminate all the instances that had an empty critic's name (2722 rows).

```
reviews = reviews[-which(reviews$critic == ""),]
reviews$critic <- droplevels(reviews$critic)
```

We also transformed the date to a ready-to-read format.

```
# this is needed to convert the date
# don't worry, your enviroment settings are preserved
lc_time <- Sys.getlocale("LC_TIME")
Sys.setlocale("LC_TIME","C")
reviews$date = as.Date(reviews$date, "%B %d, %Y")
Sys.setlocale("LC_TIME", lc_time)
```

Subsequently, the ratings presented a very high number of different levels, as these come from different sites and have different scales. We converted all ratings to the 1-5 scale. Note that we can only convert ratings when we are sure about their scale. Because of that, we chose only to convert the ratings which represented a division (numerator/denominator) and the ratings which had letters A through F (except E). We considered the A-F ratings to be a simple 1-5 scale (A is 5, B is 4, and so on). The instances that could not be translated to the 1-5 scale were transformed into NA.

```
NAs in attribute "rating" after preprocessing:


FALSE  TRUE
27010 24700
```

The reader should be aware that there were some critics that rated the same movie multiple times. To deal with that, we simply took the rounded mean of such ratings.

```
tmp = reviews %>% group_by(critic, id) %>%
        summarise(rating=round(mean(rating))) %>% data.frame()
reviews = reviews %>% select(-rating) %>% merge(tmp, by = c("critic", "id"))
reviews = reviews[!duplicated(reviews[,c('critic','id')]),]
```

Finally, the ratings matrices could be created. We needed both a binary and a real ratings matrix, as we want to test the quality of both systems. To create the matrices, we used two filters in order to consider only the movies and the critics that appear more than a certain number of times.

To create the real ratings matrix, the instances that had missing values in the ratings column were omitted as we couldn't be sure of what the recommender would do with those instances. By omitting them we know that the recommender will consider that the user didn't see the movies in question, which is the desired effect since we want to, in theory, predict what rating a user will give to a movie.

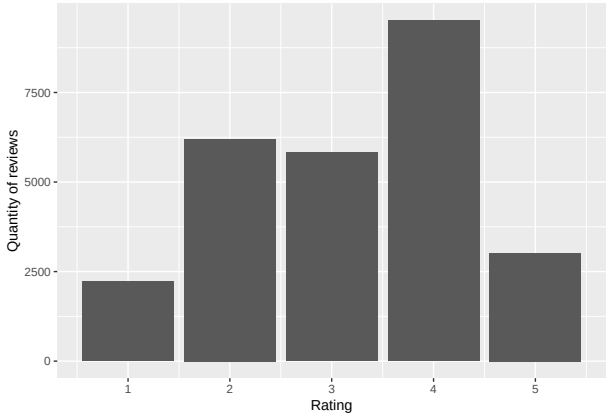
```
min_ratings <- function(data, filt1=5, filt2=5){

  a = data %>% group_by(id) %>% summarise(n = n())
  a = a %>% filter(n > filt1)
  data = data %>% filter(id %in% a$id)

  if(nrow(data)==0) return(data)

  a = data %>% group_by(critic) %>% summarise(n = n())
  a = a %>% filter(n > filt2)
  data = data %>% filter(critic %in% a$critic)

  return(data)
}

data = min_ratings(reviews) %>% select('critic', 'id', 'rating')

binary <- as(data, "binaryRatingMatrix")


data = data %>% na.omit()

real <- as(data, "realRatingMatrix")
```

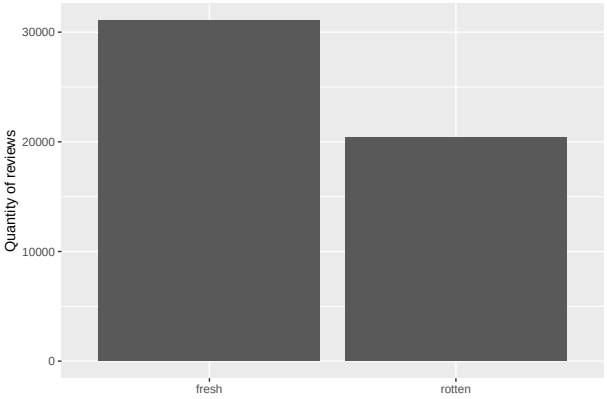
Note that we don't use the matrices created here throughout the work, as we built the needed matrices on-the-fly, giving us the opportunity to vary the filters' threshold if needed. Despite these matrices serving only for demonstration and visualization, we in fact use, almost exclusively, these thresholds (more than 5 appearances for movies and for critics) to report our findings.

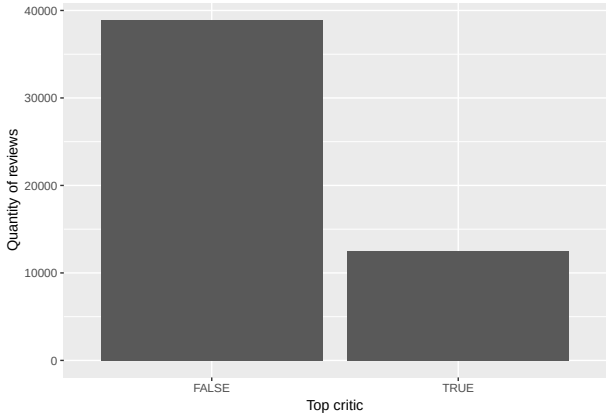
Data Visualization

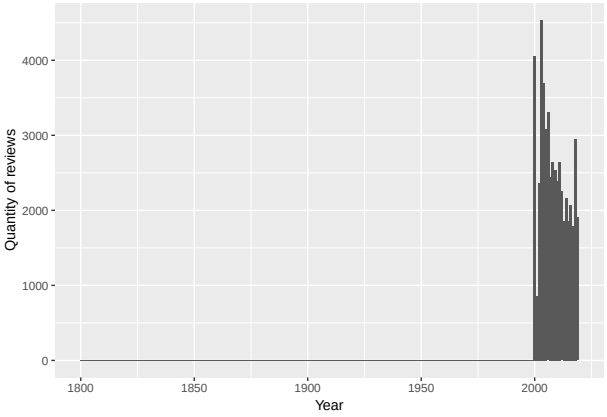
In this section we visualize both the data after all the preprocessing done and the resulting binary/real matrix.

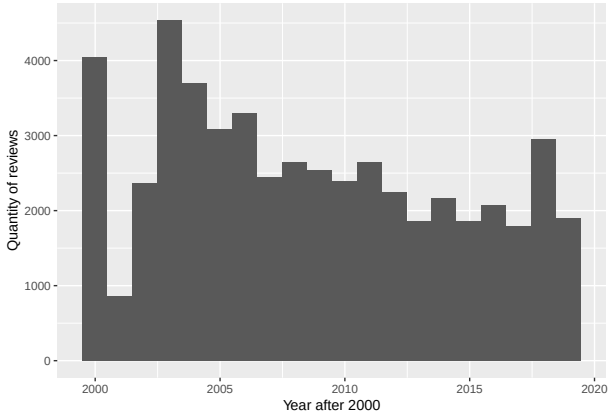
Data after preprocessing









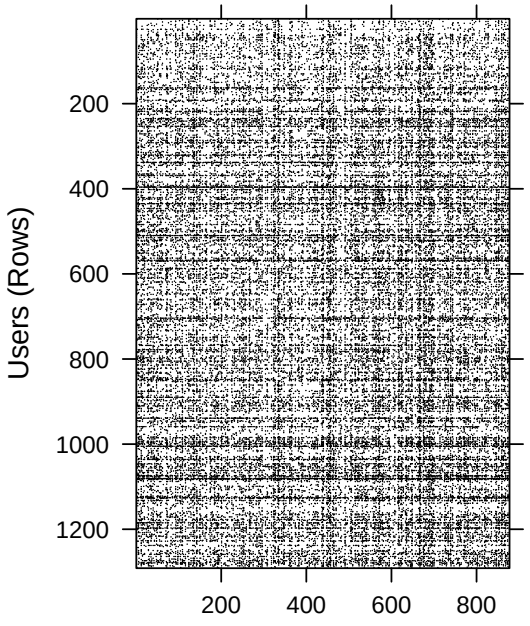



```
Statistical summary of attribute "date":

       Min.      1st Qu.       Median         Mean      3rd Qu.
"1800-01-01" "2003-09-23" "2007-11-20" "2008-08-06" "2013-05-25"
       Max.
"2018-12-06"


Number of different publishers:  1282
```

```
Top publishers:


eFilmCritic.com EmanuelLevy.Com Washington Post
            631             576             565


Number of different critics:  3496

Top critics:


   Emanuel Levy     Roger Ebert Dennis Schwartz
            579             461             413
```

Binary ratings matrix

```
Movies with least appearances:

 490  968  744 1861 1942  288
   1    1    3    4    4    5


Movies with most appearances:

 782 1777 1525 1083 1704 1071
 251  230  228  226  225  224


Users with least appearances:

Abbie Bernstein  Adam Fendelman   Alan Morrison
              6               6               6


Users with most appearances:

      Emanuel Levy        Roger Ebert James Berardinelli
               464                449                347


Number of different users (rows):  1291

Number of different movies (columns):  877
```

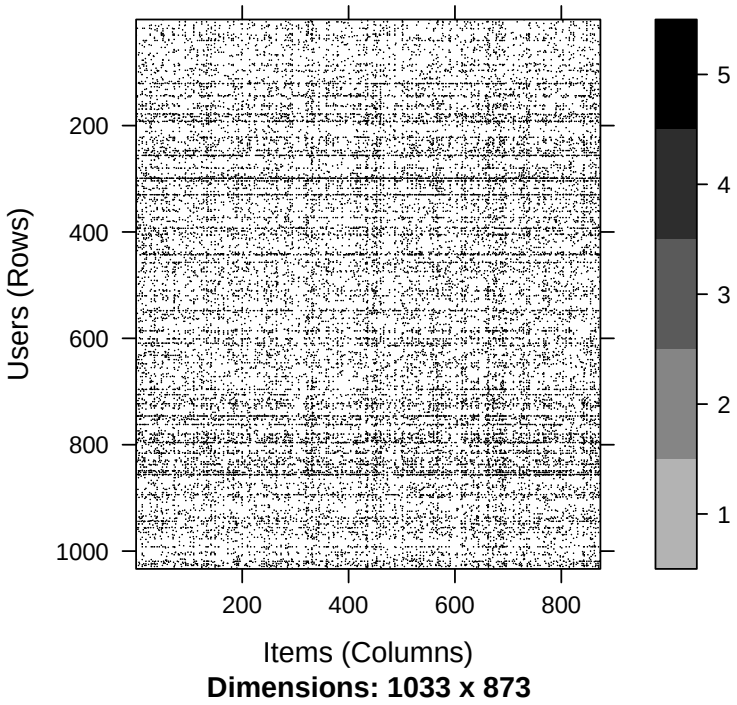


Dimensions: 1291 x 877

Real ratings matrix

```
Movies with least appearances:

 895 1264 1331 1391 1942   44
   1    1    1    1    1    2


Movies with most appearances:

1071 1083 1136  782 1777 1017
 125  121  121  119  115  113


Users with least appearances:

Alison Willmore     Anita Gates  Anita Schmaltz
              1               1               1


Users with most appearances:

   Emanuel Levy   Frank Swietek Dennis Schwartz
            463             344             335


Number of different users (rows):  1033
```

```
Number of different movies (columns):  873
```



Dimensions: 1033 x 873

Modelling approaches

In this section, we describe the methods used to obtain recommendations. We also show them in action using the 50 critics with most reviews – 49 to train, 1 to predict – and the movies with at least 5 appearances among these critics.

We chose not to include the test critic (chosen randomly, Peter Bradshaw) in the training dataset because this is how a real system would work. We do not evaluate the results for the test critic, we merely want to show how the methods work.

Recommender systems can be based in binary ratings – the critic reviewed or not a movie – or real ratings – the score the critic gave a movie. The methods for binary ratings recommend the movies the user is likely to review while the methods for real ratings try to predict the ratings the user will give to different movies and recommend the ones with the highest predicted rating.

We show only the methods for the binary ratings because the methods for real ratings are analogous.

```
data = reviews %>% select(critic, id, rating)
a = data %>% group_by(critic) %>% summarise(n = n()) %>% arrange(-n)
a = a %>% top_n(50, n)
data = data %>% filter(critic %in% a$critic)

a = data %>% group_by(id) %>% summarise(n = n())
a = a %>% filter(n > 5)
data = data %>% filter(id %in% a$id)

crit = "Peter Bradshaw"
```

Movies already seen by Peter Bradshaw:

```
data %>% filter(critic == crit) %>% select(id) %>% pull %>% sort
```

```
  [1]    3   10   13   23   54   57   65   77   95  108  110  133  136  140
 [15]  153  205  210  218  227  235  251  267  274  290  300  304  321  322
 [29]  338  368  377  394  398  410  427  453  456  458  482  486  488  524
 [43]  564  582  590  596  610  661  667  672  674  689  712  714  740  756
 [57]  760  769  782  791  797  809  815  845  848  850  864  903  906  911
 [71]  930  953  957  973 1009 1011 1014 1017 1028 1044 1050 1067 1071 1083
 [85] 1084 1092 1103 1112 1136 1163 1200 1231 1235 1236 1305 1307 1320 1321
 [99] 1376 1399 1415 1418 1420 1424 1442 1447 1450 1466 1475 1483 1486 1488
[113] 1494 1512 1520 1523 1528 1545 1561 1564 1574 1579 1593 1598 1606 1646
[127] 1682 1704 1744 1757 1762 1775 1777 1788 1801 1808 1822 1832 1837 1857
[141] 1877 1882 1892 1903 1907 1929 1931 1944 1968 1976 1995 1996
```

```
bm = as(data, "binaryRatingMatrix")
bm.train <- bm[-which(bm@data@itemsetInfo == crit),]
peter <- bm[which(bm@data@itemsetInfo == crit),]
```

We first show the popular method. This method works by determining the most popular movies among the critics in the training dataset and recommending the movies that appear the most. If the user we are recommending to has already seen a movie, it is removed from the recommendations.

```
m1 <- Recommender(bm.train, "POPULAR")
pred1 <- predict(m1, peter, n = 5)
```

10 most popular movies:

```
m1@model$topN@itemLabels[m1@model$topN@items[[1]]][1:10]
```

```
 [1] "976"  "1071" "1877" "1844" "940"  "1183" "1585" "1777" "218"  "1442"
```

Movies recommended to Peter Bradshaw by POPULAR:

```
getList(pred1)[[1]]
```

```
[1] "976"  "1844" "940"  "1183" "1585"
```

In our example, the most popular movie is “976” and since the user hasn’t seen this movie yet, the algorithm recommends it. The second most popular movie is “1071” but the user has already seen it so it is not recommended. The other 4 recommendations are obtained in the same manner.

We move on to the association rules method. With the discovered rules, the method recommends movies that are on the right hand side of rules that have on the left hand side movies reviewed by the user. We use a low support – $10/|DB|$. 3

```
m2 <- Recommender(bm.train, "AR", parameter = list(support = 0.2))
rule<-getModel(m2)
```

Example rule:

```
inspect(rule$rule_base[1,])
```

```
    lhs      rhs    support   confidence lift     count
[1] {908} => {1183} 0.2156863 1          1.342105 11
```

```
pred <- predict(m2, peter, n = 5)
```

```
Movies recommended to Peter Bradshaw by AR:
```

```
getList(pred)[[1]]
```

```
[1] "1777" "158"  "458"  "1788" "1376"
```

Above we present a rule as an example. We noticed that this method recommends movies that the user has already reviewed. We were unable to identify the problem so this method will probably give irrelevant recommendations throughout this project.

The last method is collaborative filtering (CF). There are two types of CF: user based and item based.

User based collaborative filtering computes the similarity between critics and then uses the movies reviewed by the k nearest critics to make recommendations for a user. Similariy between critics is computed based on the movies they reviewed.

Item based collaborative filtering, on the other hand, computes the similarity between movies and uses the k nearest movies to make recommendations for a user. Similarity between movies is based on the critics that reviewed them.

To demonstrate with the reviews dataset, we chose the cosine similarity as the similarity functions.

We present only the movies that each method recommends since the similarity matrices are too big to show.

```
m3 <- Recommender(bm.train,"IBCF",parameter=list(method = "cosine"))
pred <- recommenderlab::predict(m3,peter,n=5)
```

```
Movies recommended to Peter Bradshaw by IBCF:
```

```
getList(pred)[[1]]
```

```
[1] "976"  "1844" "301"  "1183" "158"
```

```
m4 <- Recommender(bm.train,"UBCF",parameter=list(method = "cosine"))
pred <- recommenderlab::predict(m4,peter,n=5)
```

```
Movies recommended to Peter Bradshaw by UBCF:
```

```
getList(pred)[[1]]
```

```
[1] "158"  "1588" "1844" "976"  "1685"
```

As can be verified, only the association rules method gives movie recommendations that the user has already seen. We also note that some movies are recommended by a lot of methods which can mean that these are good recommendations.

Recommender System

In this section, we evaluate, for binary and real ratings, the methods explained in the section "Modelling approaches". For that, we implemented some functions to help automate this process:

- evaluation – given a ratings matrix, the evaluation protocol and the type of matrix, this function will evaluate a pre-chosen list of models and will then return the respective metrics for each of them.
- recommend – this procedure will first filter the data and then use the function above with 3 different evaluation protocols (all but one, given 2 and given 4) and return a dataframe with the adequate metrics for each of the protocols.
- context – this function does the same as the "recommend" function above but instead of creating recommendation systems for only 1 matrix, it evaluates the models for multiple matrices related to different categories of a given context. This procedure will then average the metrics of all categories that had more than 250 ratings.

We tested each method with different parameters to see which produced the best results. To avoid comparing 10-18 different models, we selected the best model within each method to be representative of that method. This means that despite having, for example, 5 different association rules models, only one of these will be chosen per protocol to be visualized. The best model was chosen based on precision for binary ratings and based on RMSE for real ratings. We chose precision because we think that it's more important to be sure that we're recommending a movie that the user will like than to recommend all the movies that the user will like.

```
evaluation <- function(data, given = -1, type = "binary", debug = F){

  algorithms <- list(
    "Popular" = list(name = "POPULAR", parameters = NULL),
    "AR1" = list(name = "AR", parameters = NULL),
    "AR2" = list(name = "AR", parameters = list(support = 0.05)),
    "AR3" = list(name = "AR", parameters = list(support = 0.15)),
    "AR4" = list(name = "AR", parameters = list(confidence = 1)),
    "AR5" = list(name = "AR", parameters = list(confidence = 0.5)),
    "UBCF1" = list(name = "UBCF", parameters = NULL),
    "UBCF2" = list(name = "UBCF", parameters = list(method = "cosine")),
    "UBCF3" = list(name = "UBCF", parameters = list(method = "cosine", weighted = F)),
    "UBCF4" = list(name = "UBCF", parameters = list(method = "cosine", nn = 11)),
    "UBCF5" = list(name = "UBCF", parameters = list(method = "cosine", nn = 51)),
    "IBCF1" = list(name = "IBCF", parameters = NULL),
    "IBCF2" = list(name = "IBCF", parameters = list(method = "cosine")),
    "IBCF3" = list(name = "IBCF", parameters = list(method = "cosine", normalize_sim_matrix = T)),
    "IBCF4" = list(name = "IBCF", parameters = list(method = "cosine", k = 11)),
    "IBCF5" = list(name = "IBCF", parameters = list(method = "cosine", k = 51)),
    "IBCF6" = list(name = "IBCF", parameters = list(method = "cosine", alpha = 0.1)),
    "IBCF7" = list(name = "IBCF", parameters = list(method = "cosine", alpha = 0.9))
  )

  algorithms_real <- list(
  "Popular" = list(name = "POPULAR", parameters = NULL),
  "UBCF1" = list(name = "UBCF", parameters = NULL),
  "UBCF2" = list(name = "UBCF", parameters = list(nn = 11)),
  "UBCF3" = list(name = "UBCF", parameters = list(nn = 51)),
  "IBCF1" = list(name = "IBCF", parameters = NULL),
  "IBCF2" = list(name = "IBCF", parameters = list(normalize_sim_matrix = T)),
```

```
"IBCF3" = list(name = "IBCF", parameters = list(k = 11)),
"IBCF4" = list(name = "IBCF", parameters = list(k = 51)),
"IBCF5" = list(name = "IBCF", parameters = list(alpha = 0.1)),
"IBCF6" = list(name = "IBCF", parameters = list(alpha = 0.9))
)


if(type == "binary"){
  m <- as(data, "binaryRatingMatrix")
  scheme <- m %>% evaluationScheme(given = given)
  results <- recommenderlab::evaluate(scheme, algorithms, n = c(1,2,5), progress = F)

  if(length(results@.Data) == 0) return(NULL)

  results_tbl <- data.frame(name=NA,n=NA,
                            method=NA,
                            TP=0, FP=0,
                            FN=0, TN=0,
                            precision=0,
                            recall=0,
                            stringsAsFactors=FALSE)
  for(i in 1:length(results@.Data)){
   for(j in 1:3){
      results_tbl[(i-1)*3+j,1:3] <- c(names(results)[i],
                                      rownames(results@.Data[[i]]@results[[1]]@cm)[j],
                                      results[[i]]@method)
      results_tbl[(i-1)*3+j,4:length(results_tbl)] <- results@.Data[[i]]@results[[1]]@cm[j,]
    }
  }
}

else{
  m <- as(data, "realRatingMatrix")
  scheme <- m %>% evaluationScheme(given = given, goodRating = 3)
  results <- recommenderlab::evaluate(scheme, algorithms_real, type = "ratings", progress = F)

  if(length(results@.Data) == 0) return(NULL)

  results_tbl <- data.frame(name=NA, n = NA,
                            method=NA,
                            RMSE=0, MSE=0,
                            MAE=0,
                            stringsAsFactors=FALSE)
  for(i in 1:length(results@.Data)){
    results_tbl[i,1:3] <- c(names(results)[i],
                                    NA,
                                    results[[i]]@method)
    results_tbl[i,4:length(results_tbl)] <- results@.Data[[i]]@results[[1]]@cm[1,]
  }
}

return(results_tbl)
```

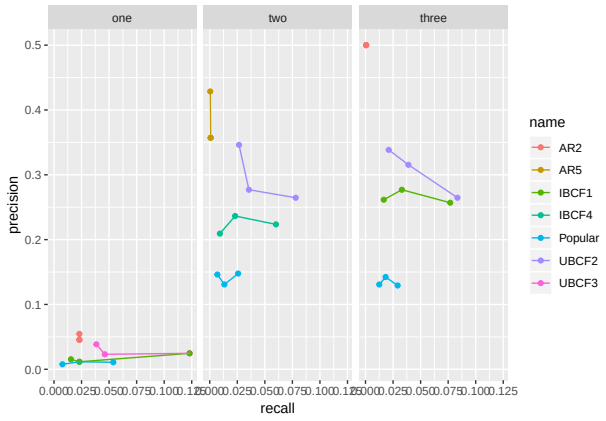
```
}
```

Binary Predictions

The models used to build a recommender system for binary ratings can be seen in the function *evaluation*. We present the results of applying these models to the whole binary ratings matrix.

The AR method fails for some parameters since there are no rules with the given combination of minimum support and minimum confidence.

```
data = reviews %>% select(critic, id)
r = recommend(data)
```



We note that not all protocols choose the same model for each method. We note also that the protocols two and three give in general better precision but worse recall than protocol one. This is due to the fact that protocols two and three predict with only two and four known ratings, respectively, which means that they have more chances of hitting the remaining ratings when they make 1, 2 and 5 predictions. This also means that the recall will be closer to zero because there will be a lot of ratings that the algorithm will not be able to predict, as it can only give up to 5 recommendations when there are sometimes hundreds of possible, correct recommendations.

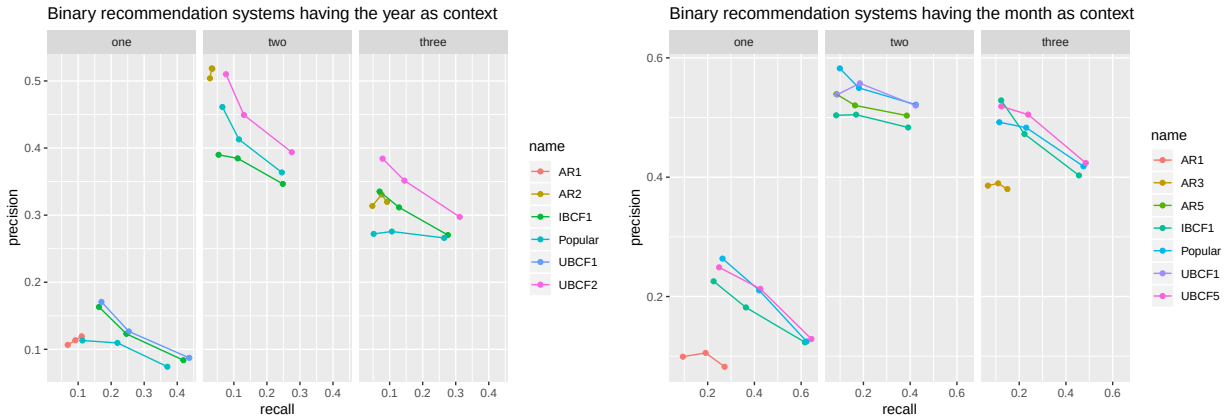
In order to use the review date information in our recommendations, we attempted to use the first 90% of reviews for training and the remaining for testing. We also attempted to implement a sliding window and an expanding window to make recommendations. These approaches failed due to the fact that the *evaluationScheme* function always randomly samples the data when creating the training and testing sets.

We decided, therefore, to separate the data by year of review and average the results for each year. This was done using the *context* function. We also separated the data by month to see if we obtained better results.

As before, the AR method fails at times, so we removed from the results the models that failed for at least one category. This is true for every use of the *context* function.

```
data = reviews %>% select(critic, id, date)
d = split(data, year(data$date))
r1 = context(d)
```

```
data = reviews %>% select(critic, id, date)
d = split(data, month(data$date))
r = context(d)
```

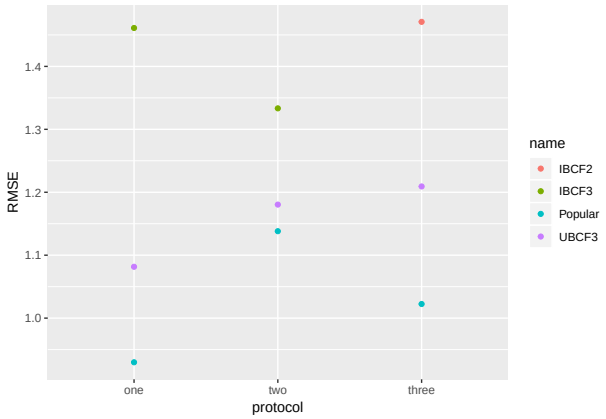


Both the division per year and per month show an increase in precision and recall in the first protocol and an increase in recall in the second and third protocols when compared to the whole dataset.

Real Predictions

The models used to build a recommender system for real ratings can be seen in the function *evaluation*. We present the results of applying these models to the whole real ratings matrix.

```
data = reviews %>% select(critic, id, rating) %>% na.omit()
r = recommend(data, "real")
```

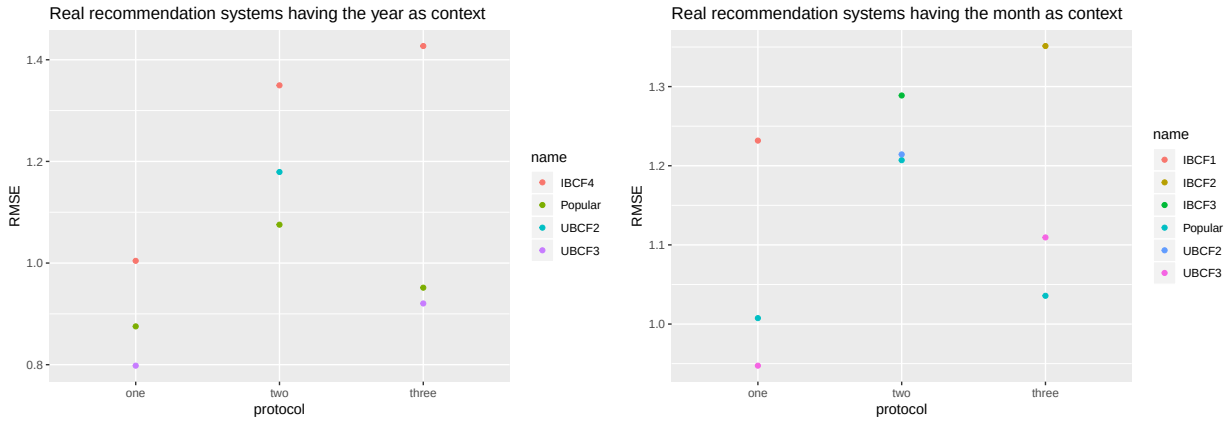


The lowest RMSE is achieved with the first protocol. This is likely due to the fact that in this protocol all but one rating is used to predict a single rating, while in the other protocols the number of ratings used is lower.

We also split the data by year and then by month to see if it improved the RMSE.

```
data = reviews %>% select(critic, id, rating, date) %>% na.omit()
d = split(data, year(data$date))
r1 = context(d, "real")
```

```
data = reviews %>% select(critic, id, rating, date) %>% na.omit()
d = split(data, month(data$date))
r = context(d, "real")
```



The results of the divided data are quite similar to the results for the whole data so we consider there isn't an improvement in RMSE.

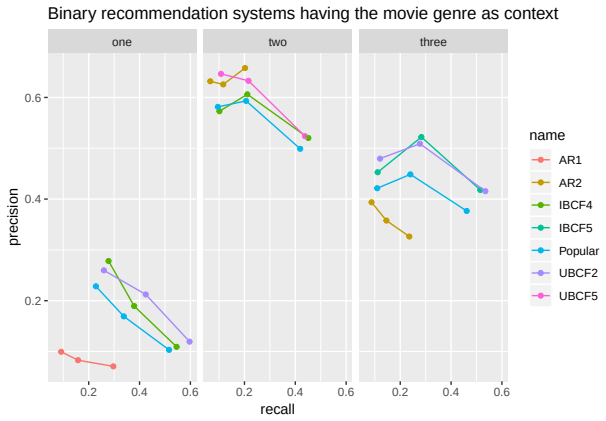
Context-aware recommendation systems

Imagine that a user is in the drama section of, say, Netflix. In this case, it wouldn't make sense to recommend movies that don't relate to that genre. In other words, context is important because a user might be expecting to receive recommendations that relate to some context that he might be interested in.

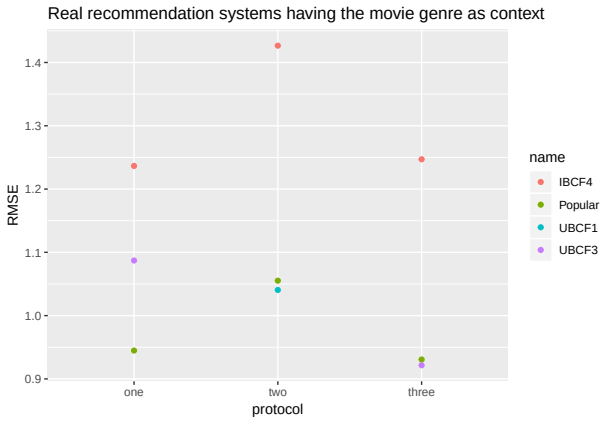
There are multiple ways to do context-aware recommendation systems. We chose to simply divide the dataset according to the chosen context and produce a recommendation system to that subset of data.

To test the power of context in this dataset, we chose two attributes – genre and studio. We then used the *context* function mentioned in the last section in order to have an average of how the algorithms behave in the different contexts of each attribute. Note that these metrics vary for each category and it can be useful to look at the metrics of a model in a single category rather than the average of an attribute.

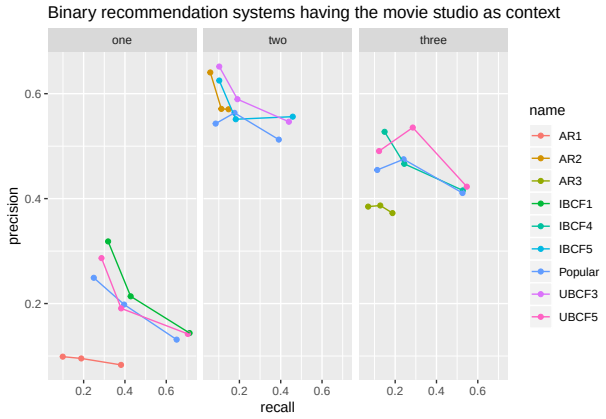
```
big_data = reviews %>% merge(movies, by = "id")

data = big_data %>% select(critic, id, genre)
d = split(data, data$genre)
r = context(d)
```



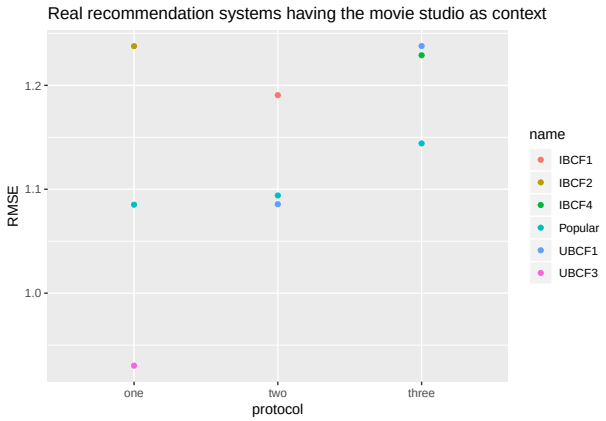
```
data = big_data %>% select(critic, id, rating.x, genre) %>% na.omit()
d = split(data, data$genre)
r = context(d, "real")
```




```
data = big_data %>% select(critic, id, studio)
data = data[-which(data$studio == ""),]
d = split(data, data$studio)
r = context(d)
```




```
data = big_data %>% select(critic, id, rating.x, studio) %>% na.omit()
data = data[-which(data$studio == ""),]
d = split(data, data$studio)
r = context(d, "real")
```



As can be see in the figures above, the context-aware recomendation systems achieve a better score than the systems that consider no context at all for binary ratings. The real ratings don't show a significant difference in RMSE compared to the systems with no context.

Conclusions

In this work, we managed to, after data preprocessing, compare the possible models for movies recommendation by the quality of their predictions, both for binary and real matrices. Given the results, we conclude that for binary ratings, a UBCF or IBCF model seems to be the best for this data. For real ratings, Popular, UBCF and IBCF seem to achieve all a similar score, so we are not able to say which is the best one. We also looked into context-aware recommendation systems, comparing models that took into account some context of the data – genre and studio of the movie – and saw what improvements that context brought to our models, concluding that it brought quality only to the binary predictions.

In future work, the reason why the AR method produced predictions that the user had already seen should assessed and taken care of. More data could also be valuable, in particular to the context-aware systems, since one could be able to create systems based on multiple contexts at the same time.