DATA-612 Recommender Systems

Project 2 : Content-Based and Collaborative

Author: Rupendra Shrestha, Bikash Bhowmik

15 Jun 2025

Contents

1 Instruction

The goal of this assignment is for to try out different ways of implementing and configuring a recommender, and to evaluate your different approaches.

For assignment 2, start with an existing dataset of user-item ratings, such as our toy books dataset, MovieLens, Jester [http://eigentaste.berkeley.edu/dataset/] or another dataset of your choosing.

Implement at least two of these recommendation algorithms:

• Content-Based Filtering • User-User Collaborative Filtering • Item-Item Collaborative Filtering

As an example of implementing a Content-Based recommender, you could build item profiles for a subset of MovieLens movies from scraping http://www.imdb.com/ or using the API at https://www.omdbapi.com/ (which has very recently instituted a small monthly fee). A more challenging method would be to pull movie summaries or reviews and apply tf-idf and/or topic modeling.

You should evaluate and compare different approaches, using different algorithms, normalization techniques, similarity methods, neighborhood sizes, etc. You don't need to be exhaustive—these are just some suggested possibilities.

You may use the course text's recommenderlab or any other library that you want. Please provide at least one graph, and a textual summary of your findings and recommendations.

2 Introduction

Recommender systems have become essential tools for guiding users toward relevant items based on preferences and past behaviors. This project explores two popular collaborative filtering techniques—Item-Based Collaborative Filtering (IBCF) and User-Based Collaborative Filtering (UBCF)—using the MovieLense dataset from the recommenderlab package in R. By evaluating these methods with various similarity metrics and tuning parameters, the goal is to compare their performance and determine effective strategies for generating personalized movie recommendations.

3 Dataset and Setup

This assignment explores the use of recommender systems using the R package recommenderlab. The vignette for this package provides useful guidance and documentation.

The MovieLense dataset contains ratings from 943 users on 1,664 movies. It can be extracted and structured as a matrix suitable for recommendation algorithms. Since many entries are missing (represented as NA or 0 in the matrix), these will be excluded during evaluation to ensure accuracy. The distribution of movie ratings is left-skewed, with most ratings falling between 3 and 5 stars—4 being the most frequent—indicating that users tend to give higher ratings, while lower scores like 1 or 2 are less common.

```r
data_package <- data(package = "recommenderlab")
data_package$results[, "Item"]
```

```
## [1] "Jester5k"                     "JesterJokes (Jester5k)"
## [3] "MSWeb"                        "MovieLense"
## [5] "MovieLenseMeta (MovieLense)" "MovieLenseUser (MovieLense)"
```

```r
data("MovieLense")

recommender_models <- recommenderRegistry$get_entries(dataType =
                                                        "realRatingMatrix")

lapply(recommender_models, "[[", "description")
```

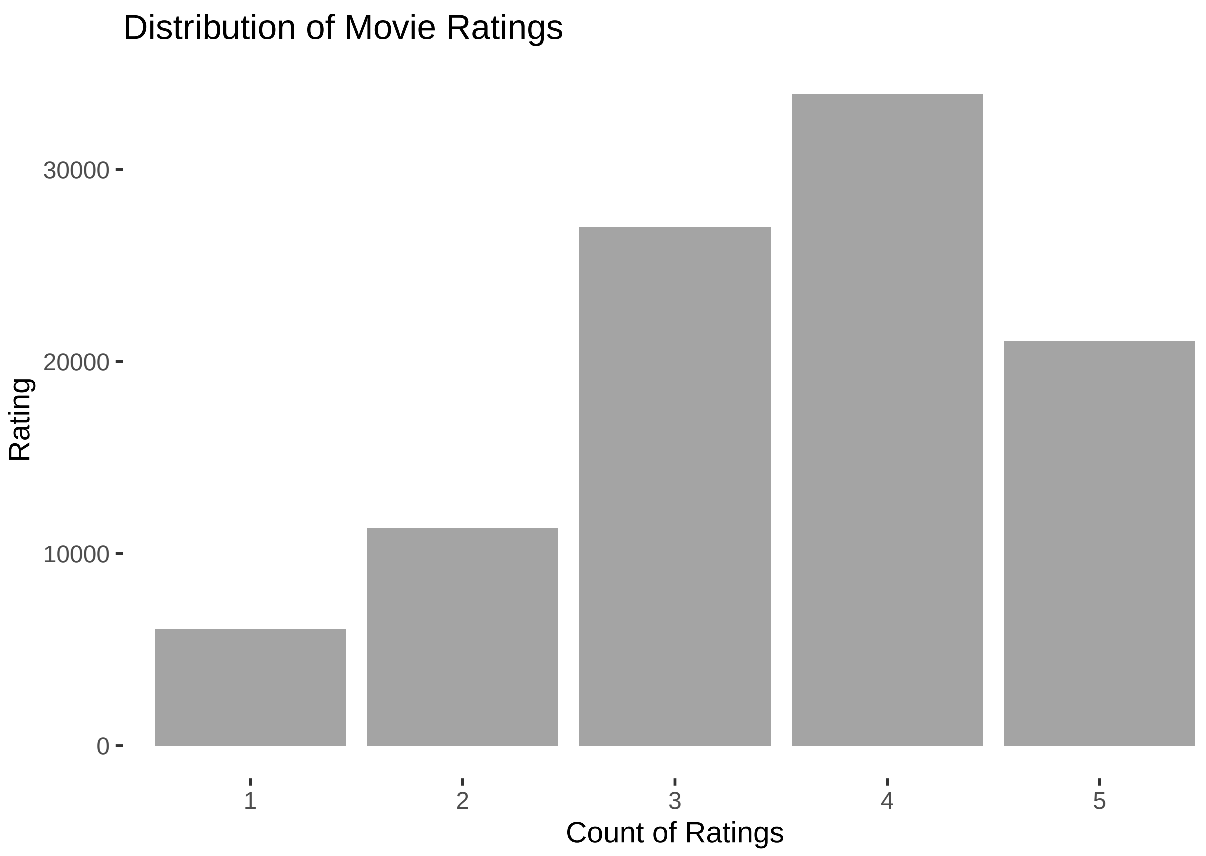
```
## $HYBRID_realRatingMatrix
## [1] "Hybrid recommender that aggegates several recommendation strategies using weighted averages."
##
## $ALS_realRatingMatrix
## [1] "Recommender for explicit ratings based on latent factors, calculated by alternating least squares algorith
##
## $ALS_implicit_realRatingMatrix
## [1] "Recommender for implicit data based on latent factors, calculated by alternating least squares algorithm."
##
## $IBCF_realRatingMatrix
## [1] "Recommender based on item-based collaborative filtering."
##
## $LIBMF_realRatingMatrix
## [1] "Matrix factorization with LIBMF via package recosystem (https://cran.r-project.org/web/packages/recosystem/
##
## $POPULAR_realRatingMatrix
## [1] "Recommender based on item popularity."
##
## $RANDOM_realRatingMatrix
## [1] "Produce random recommendations (real ratings)."
##
## $RERECOMMEND_realRatingMatrix
## [1] "Re-recommends highly rated items (real ratings)."
##
## $SVD_realRatingMatrix
## [1] "Recommender based on SVD approximation with column-mean imputation."
##
## $SVDF_realRatingMatrix
## [1] "Recommender based on Funk SVD with gradient descend (https://sifter.org/~simon/journal/20061211.html)."
```

```
##
## $UBCF_realRatingMatrix
## [1] "Recommender based on user-based collaborative filtering."
```

```
vector_ratings <- as.vector(MovieLense@data)
unique(vector_ratings)
```

```
## [1] 5 4 0 3 1 2
```

```
table_ratings <- table(vector_ratings)

vector_ratings <- vector_ratings[vector_ratings != 0]

vector_ratings <- factor(vector_ratings)

qplot(vector_ratings,alpha = 0.5) + ggtitle("Distribution of Movie Ratings")+
  theme(panel.grid = element_blank(),
        panel.background = element_blank(),
        legend.position = 'none')+
  labs(x = "Count of Ratings",
       y = "Rating")
```



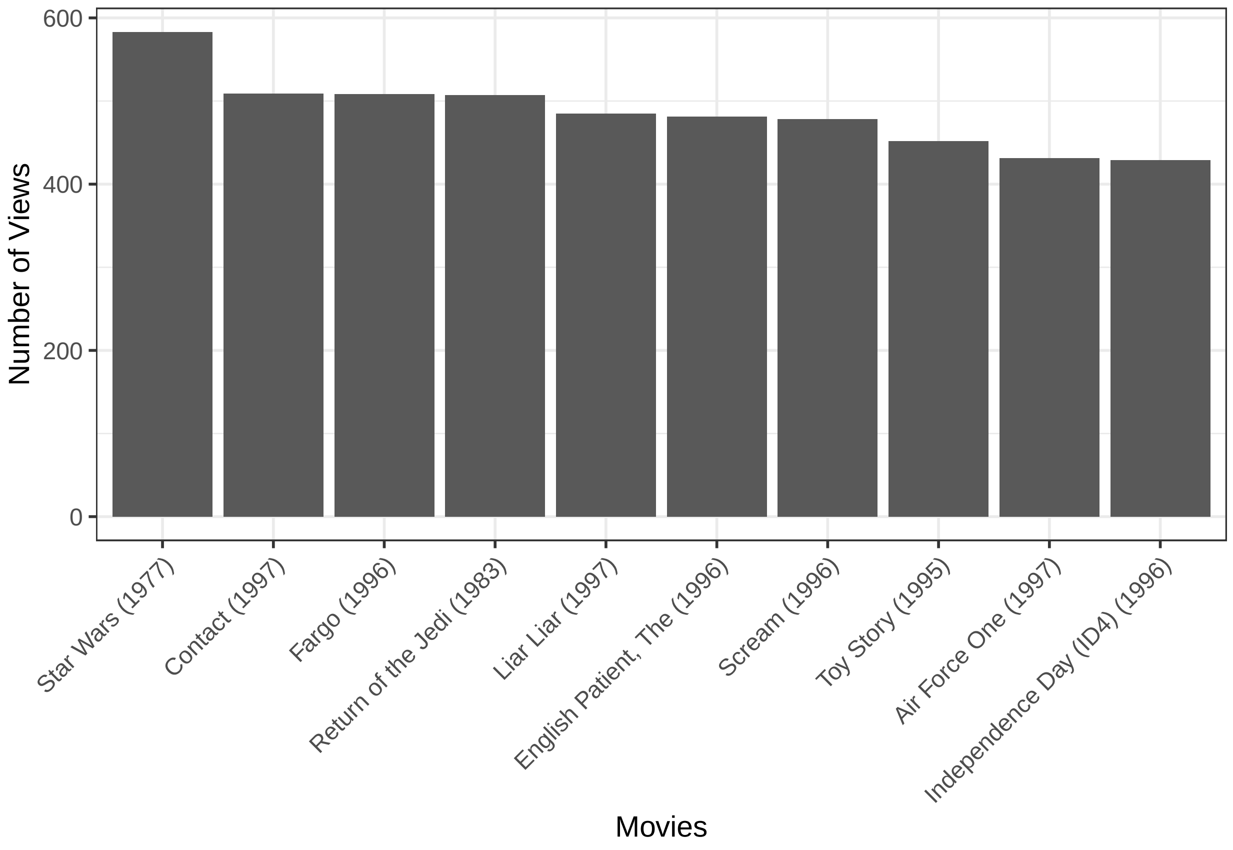
Looking deeper into the MovieLense dataset, we can look at which movies are the most viewed and popular within the dataset. The following table shows all movies within the dataset ordered by popularity. The 3 figures below highlight: -Average of movie ratings with all movies considered. -Average of movie ratings with adjusted distribution. -Average of user ratings.

We expect that the peak in 1 star/5 star(albeit small) reviews is related to movies with too few reviewers to be considered a robust rating. Thus the second plot provides this same distribution except with movies that have at least 100 reviews.

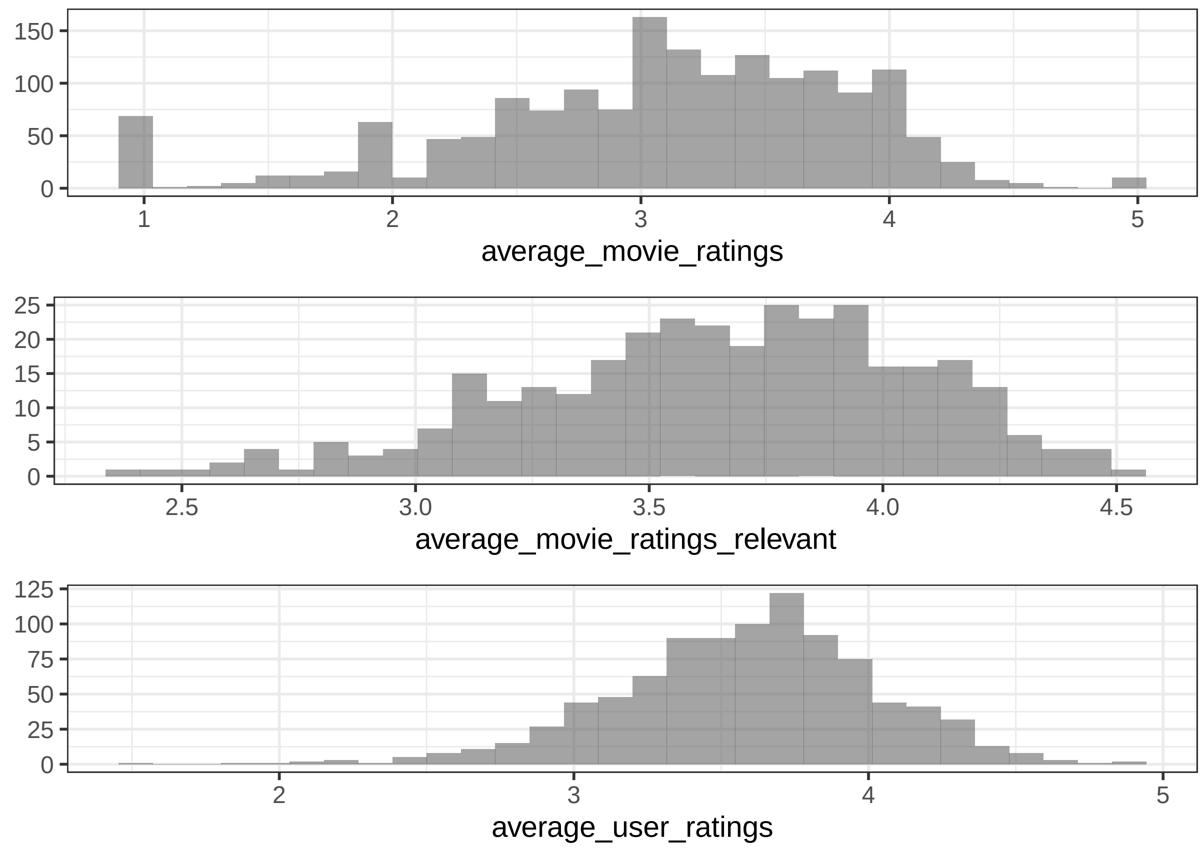
```
views_per_movie<-colCounts(MovieLense)



table_views <- data.frame(
  movie = names(views_per_movie),
  views = views_per_movie
) %>% arrange(desc(views))

table_views %>%
  head(40) %>%
  kable() %>%
  kable_styling(
    latex_options = c("HOLD_position", "striped", "condensed"),
   position = "center"
  ) %>%
  row_spec(0, bold = TRUE, color = "white", background = "#ea7872")
```

	movie	views
Star Wars (1977)	Star Wars (1977)	583
Contact (1997)	Contact (1997)	509
Fargo (1996)	Fargo (1996)	508
Return of the Jedi (1983)	Return of the Jedi (1983)	507
Liar Liar (1997)	Liar Liar (1997)	485
English Patient, The (1996)	English Patient, The (1996)	481
Scream (1996)	Scream (1996)	478
Toy Story (1995)	Toy Story (1995)	452
Air Force One (1997)	Air Force One (1997)	431
Independence Day (ID4) (1996)	Independence Day (ID4) (1996)	429
Raiders of the Lost Ark (1981)	Raiders of the Lost Ark (1981)	420
Godfather, The (1972)	Godfather, The (1972)	413
Pulp Fiction (1994)	Pulp Fiction (1994)	394
Twelve Monkeys (1995)	Twelve Monkeys (1995)	392
Silence of the Lambs, The (1991)	Silence of the Lambs, The (1991)	390
Jerry Maguire (1996)	Jerry Maguire (1996)	384
Rock, The (1996)	Rock, The (1996)	378
Empire Strikes Back, The (1980)	Empire Strikes Back, The (1980)	367
Star Trek: First Contact (1996)	Star Trek: First Contact (1996)	365
Back to the Future (1985)	Back to the Future (1985)	350
Titanic (1997)	Titanic (1997)	350
Mission: Impossible (1996)	Mission: Impossible (1996)	344
Fugitive, The (1993)	Fugitive, The (1993)	336
Indiana Jones and the Last Crusade (1989)	Indiana Jones and the Last Crusade (1989)	331
Willy Wonka and the Chocolate Factory (1971)	Willy Wonka and the Chocolate Factory (1971)	326
Princess Bride, The (1987)	Princess Bride, The (1987)	324
Forrest Gump (1994)	Forrest Gump (1994)	321
Monty Python and the Holy Grail (1974)	Monty Python and the Holy Grail (1974)	316
Saint, The (1997)	Saint, The (1997)	316
Full Monty, The (1997)	Full Monty, The (1997)	315
Men in Black (1997)	Men in Black (1997)	303
Terminator, The (1984)	Terminator, The (1984)	301
E.T. the Extra-Terrestrial (1982)	E.T. the Extra-Terrestrial (1982)	300
Dead Man Walking (1995)	Dead Man Walking (1995)	299
Leaving Las Vegas (1995)	Leaving Las Vegas (1995)	298
Schindler's List (1993)	Schindler's List (1993)	298
Braveheart (1995)	Braveheart (1995)	297
L.A. Confidential (1997)	L.A. Confidential (1997)	297
Terminator 2: Judgment Day (1991)	Terminator 2: Judgment Day (1991)	295
Conspiracy Theory (1997)	Conspiracy Theory (1997)	295

```
table_views %>%
  top_n(10,views) %>%
  ggplot(aes(x = fct_reorder(movie, -views), y = views))+
  geom_col()+
  theme_bw()+
  theme(axis.text.x = element_text(angle = 45,hjust = 1))+
  labs(x = "Movies", y = "Number of Views")
```



```
average_movie_ratings<- colMeans(MovieLense)
average_user_ratings<- rowMeans(MovieLense)

average_movie_ratings_relevant <- average_movie_ratings[views_per_movie > 100]
p1<-qplot(average_movie_ratings,alpha = 0.5)+theme_bw()+theme(legend.position = 'none')
p2<-qplot(average_movie_ratings_relevant, alpha = 0.5 )+theme_bw()+theme(legend.position = 'none')
p3<-qplot(average_user_ratings, alpha = 0.5 )+theme_bw()+theme(legend.position = 'none')


grid.arrange(p1,p2,p3, ncol = 1)
```



4 Recommender System

4.1 Setup

The recommender systems that will be evaluated in this section are Item-Based Collaborative Filtering (IBCF) and User-Based Collaborative Filtering (UBCF). "Collaborative" refers to the notion that users collaborate with each other via similarities that are defined in the models to come up with new recommendations.

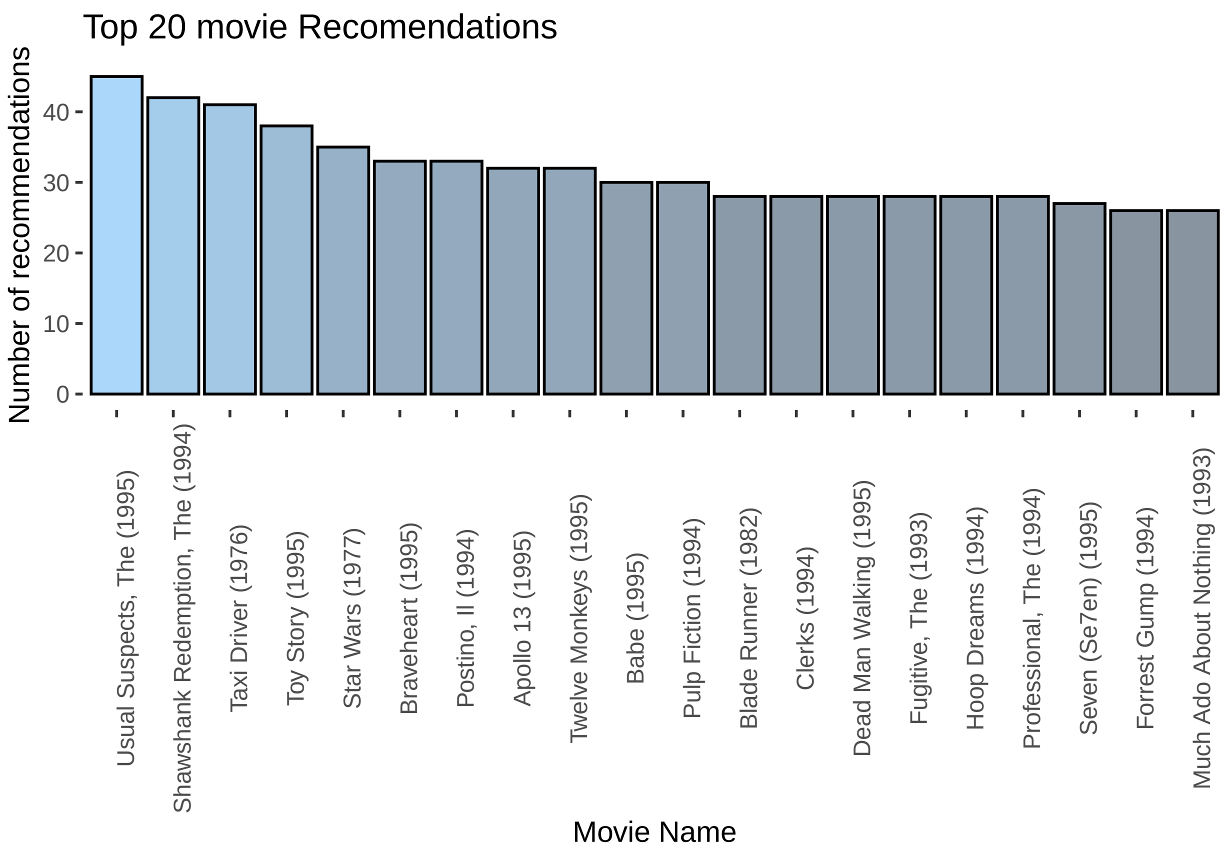
The dataset will use cross validations and will be split into train/test datasets at a 80/20 ratio. The number of recommendations made to each user will be specified as 10.

```
#filtering for users who have watched at least 50 movies, and movies with at least 100 views
ratings_movies <- MovieLense[rowCounts(MovieLense) > 50,
                             colCounts(MovieLense) > 100]



train_percent<-0.8
kept_items<-15
rating_threshold<-3
n_eval<-3
no_recommendations<-10




eval_sets <- evaluationScheme(data = ratings_movies, method = "cross-validation",
                              train = train_percent, given = kept_items, goodRating = rating_threshold, k = n_eval


#used to train
recc_train<- getData(eval_sets,'train')
#used to predict
recc_known<-getData(eval_sets,'known')
#used to test
recc_unknown<-getData(eval_sets,'unknown')
```

4.2 Item-Item Collaborative Filtering

Item-Based Collaborative Filtering (IBCF) focuses on identifying similarities between items—in this case, movies—based on user ratings. Rather than comparing users directly, the algorithm examines how similarly items are rated by various users. It then recommends movies that are most similar to those a user has already rated highly. The key steps involve computing item-to-item similarities, selecting the top k similar items, and generating personalized recommendations using the user's rating history. The following figure displays the 20 most frequently recommended movies using this approach.

```r
recc_eval<-Recommender(recc_train,method = "IBCF", parameter = NULL)

recc_predict<-predict(object = recc_eval,newdata = recc_known, n = no_recommendations, type = "ratings")


recc_predicted<-predict(object = recc_eval,newdata=recc_known,n=no_recommendations)


recc_matrix <- sapply(recc_predicted@items, function(x){
  colnames(ratings_movies)[x]
})




number_of_items<-recc_matrix %>% unlist() %>% table() %>% as.data.frame()


table_top <- data.frame("names.number_of_items_top." = number_of_items$.,
                        "number_of_items_top"= number_of_items$Freq)


table_top %>%
  top_n(20) %>%
  ggplot(mapping = aes(x=fct_reorder(names.number_of_items_top.,-as.numeric(number_of_items_top)), y = as.numeric(n
  geom_col(aes(fill = as.numeric(number_of_items_top)),color = 'black', alpha = 0.5)+
  theme(axis.text.x = element_text(angle = 90),
        legend.position = 'none',
        panel.grid = element_blank(),
        panel.background = element_blank())+
  labs(x = "Movie Name",
       y = "Number of recommendations",
       title = "Top 20 movie Recomendations")
```



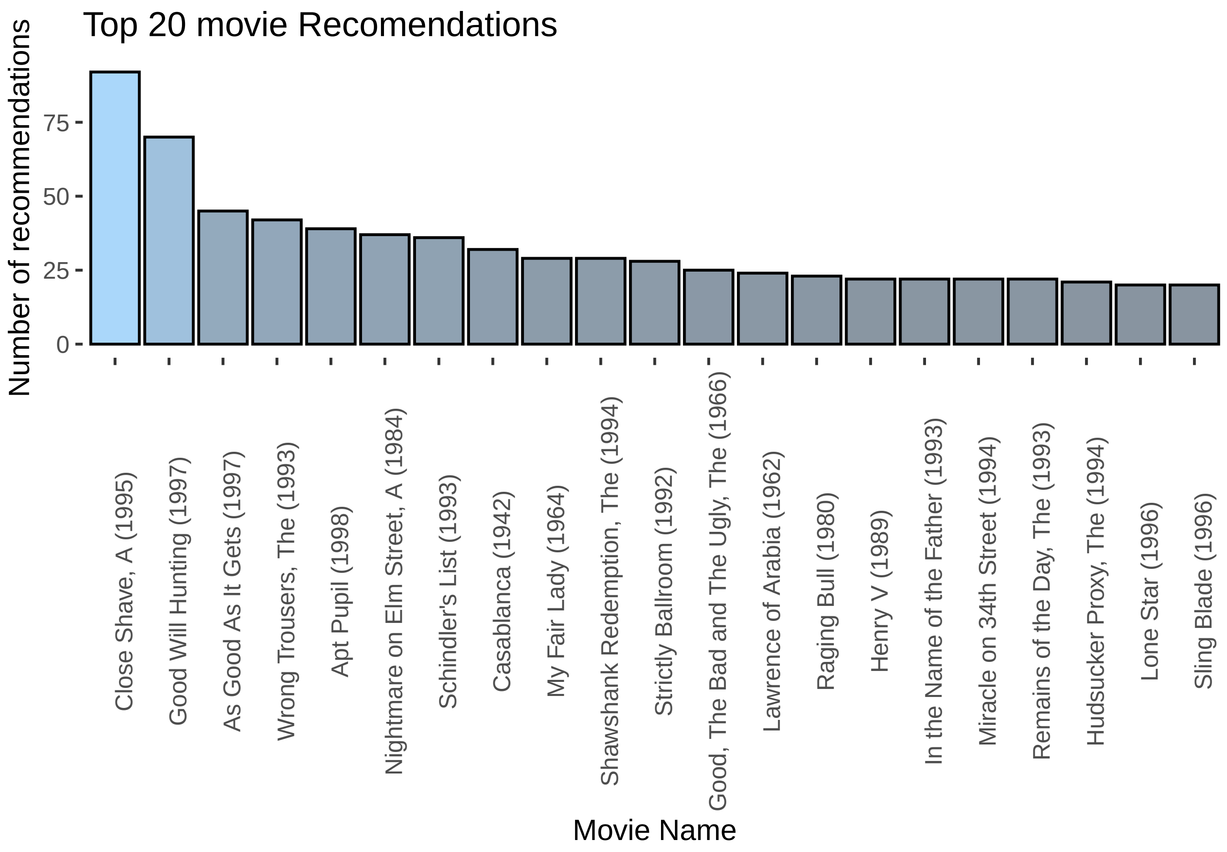
4.2.1 How IBCF Works:

Item-Based Collaborative Filtering (IBCF) creates a similarity matrix between items (movies) based on user ratings. The cosine, Pearson, or Jaccard methods are used to compute similarity scores between every pair of movies. For a target user, the algorithm looks at the movies they have rated, finds similar movies from the similarity matrix, and generates recommendations based on a weighted sum of the user's ratings multiplied by similarity scores. The top N highest-scoring items are recommended. This method is efficient when item characteristics are stable, and rating sparsity exists among users

4.3 User-User Collaborative Filtering

User-Based Collaborative Filtering (UBCF) recommends items by identifying users with similar preferences. It evaluates the similarity between users—often using techniques like k-nearest neighbors or similarity thresholds—and uses this information to suggest movies that similar users have enjoyed. Each user's past ratings serve as weighted inputs, which are adjusted based on their similarity to the target user, to prioritize recommendations. The following figure presents the top 20 movies most frequently recommended using the UBCF approach.

```r
recc_eval<-Recommender(recc_train,method = "UBCF")

recc_predict<-predict(object = recc_eval,newdata = recc_known, n = no_recommendations, type = "ratings")


#this method will use the user rating as a weight, the similarity of the movie to other movies, and multiply the w


recc_predicted<-predict(object = recc_eval,newdata=recc_known,n=no_recommendations)

recc_matrix <- sapply(recc_predicted@items, function(x){
  colnames(ratings_movies)[x]
})


number_of_items<-recc_matrix %>% unlist() %>% table() %>% as.data.frame()

table_top <- data.frame("names.number_of_items_top." = number_of_items$.,
                         "number_of_items_top"= number_of_items$Freq)


table_top %>%
  top_n(20) %>%
  ggplot(mapping = aes(x=fct_reorder(names.number_of_items_top.,-as.numeric(number_of_items_top)), y = as.numeric(n
  geom_col(aes(fill = as.numeric(number_of_items_top)),color = 'black', alpha = 0.5)+
  theme(axis.text.x = element_text(angle = 90),
        legend.position = 'none',
        panel.grid = element_blank(),
        panel.background = element_blank())+
  labs(x = "Movie Name",
       y = "Number of recommendations",
       title = "Top 20 movie Recomendations")
```



4.3.1 How UBCF Works:

User-Based Collaborative Filtering (UBCF) identifies users similar to the target user by comparing their rating vectors. The similarity (using cosine, Pearson, or Jaccard) between users is calculated, forming a user-user similarity matrix. Once similar users (neighbors) are identified, the algorithm aggregates their ratings (weighted by similarity) to predict the target user's preferences for unrated items. This method is flexible but can struggle when user overlap is low or when many users rate very few items.

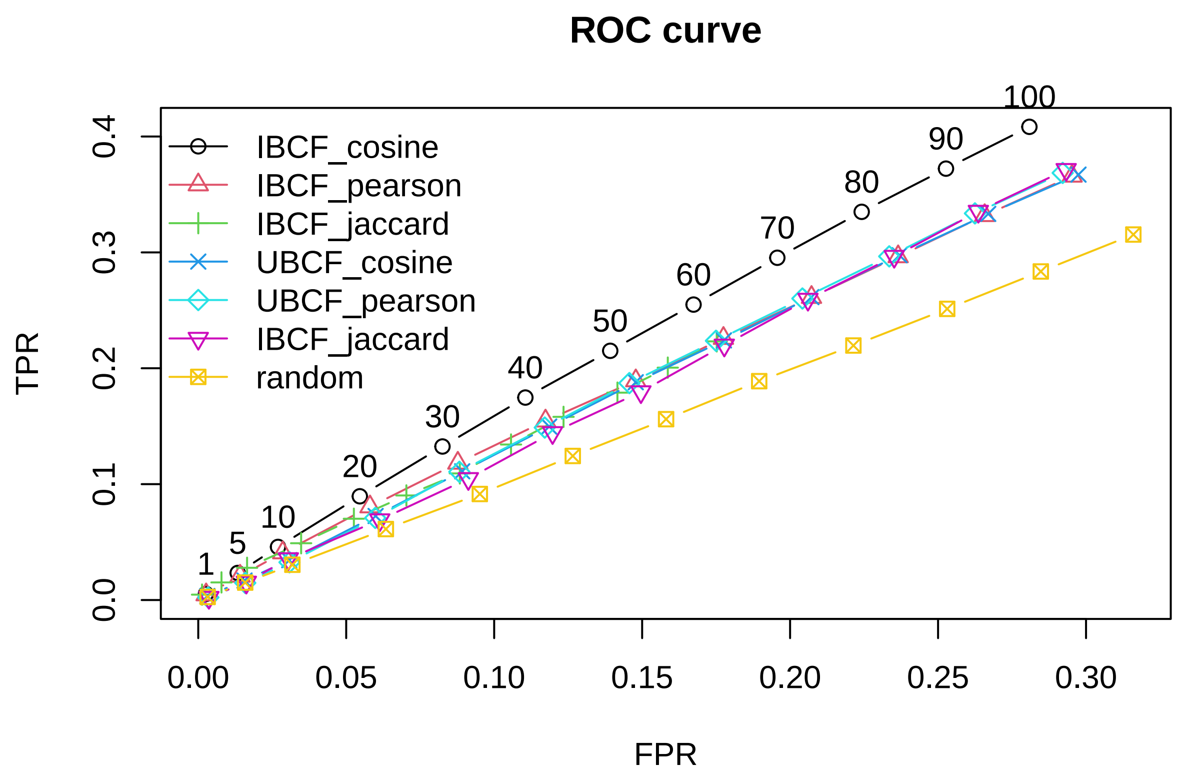
4.4 Evaluating Recommenders

There are many different recommender systems and there are several methods to define which models produce greater results. Some common metrics for model evaluations are: root mean square error (RMSE), mean square error (MSE), mean absolute error (MAE), receiver operating characteristic (ROC), area under the ROC curve (AUC). The metric that will be presented here is the ROC curve for all models (IBCF and UBCF) for each model, three methods will be evaluated, each measure the similarity between two vectors through different techniques. These include Cosine similarity function, pearson similarity function, and jaccard similarity function. In addition to these, a random or "guessing" recommendation will be produced in order to have a baseline to compare model performance to. The plot below shows the results of this evaluation and the IBCF consistently outperforms the UBCF models and the jaccard IBCF is our best.

```
models_to_evaluate <- list(
    IBCF_cosine = list(name = "IBCF", param = list(method =
                                              "cosine")),
    IBCF_pearson = list(name = "IBCF", param = list(method =
                                              "pearson")),
    IBCF_jaccard = list(name = "IBCF", param = list(method =
                                              "jaccard")),
    UBCF_cosine = list(name = "UBCF", param = list(method =
                                              "cosine")),
    UBCF_pearson = list(name = "UBCF", param = list(method =
                                              "pearson")),
    IBCF_jaccard = list(name = "UBCF", param = list(method =
                                              "jaccard")),
    random = list(name = "RANDOM", param=NULL)
  )

  n_recommendations <- c(1, 5, seq(10, 100, 10))

  list_results <- evaluate(x = eval_sets, method = models_to_evaluate, n= n_recommendations)
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.18sec/0.03sec]
##   2  [0.17sec/0.04sec]
##   3  [0.16sec/0.04sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.22sec/0.06sec]
##   2  [0.37sec/0.03sec]
##   3  [0.21sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.18sec/0.03sec]
##   2  [0.17sec/0.05sec]
##   3  [0.17sec/0.05sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.01sec/0.42sec]
##   2  [0sec/0.42sec]
##   3  [0sec/0.42sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.4sec]
##   2  [0sec/0.42sec]
##   3  [0sec/0.42sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.43sec]
##   2  [0sec/0.44sec]
##   3  [0sec/0.43sec]
## RANDOM run fold/sample [model time/prediction time]
##   1  [0sec/0.04sec]
##   2  [0.02sec/0.03sec]
##   3  [0.01sec/0.04sec]
```

```
avg_matrices <- lapply(list_results, avg)

  plot(list_results, annotate = 1, legend = "topleft")
  title("ROC curve")
```



Finally, after comparing the IBCF and UBCF performance by methods, we can dive deeper into our best performing model (IBCF-Jaccard) and try to optimize this by varying metric within the model itself. The metric that is being varied in this scenario will be k, the number of close items for the model to consider. We can iterate this over different number of recommendations. The ouput shows that there is very little improvement by varying k. Our best performing model was k = 70 but this seems negligible.

```r
vector_k <- c(1,5,seq(10,100,20))


  models_to_evaluate <- lapply(vector_k, function(k){
    list(name = "IBCF", param = list(method = "jaccard", k = k))
  })
  names(models_to_evaluate) <- paste0("IBCF_k_", vector_k)

  n_recommendations <- c(1, 5, seq(10, 100, 10))
  list_results <- evaluate(x = eval_sets, method = models_to_evaluate, n
                           = n_recommendations)
```

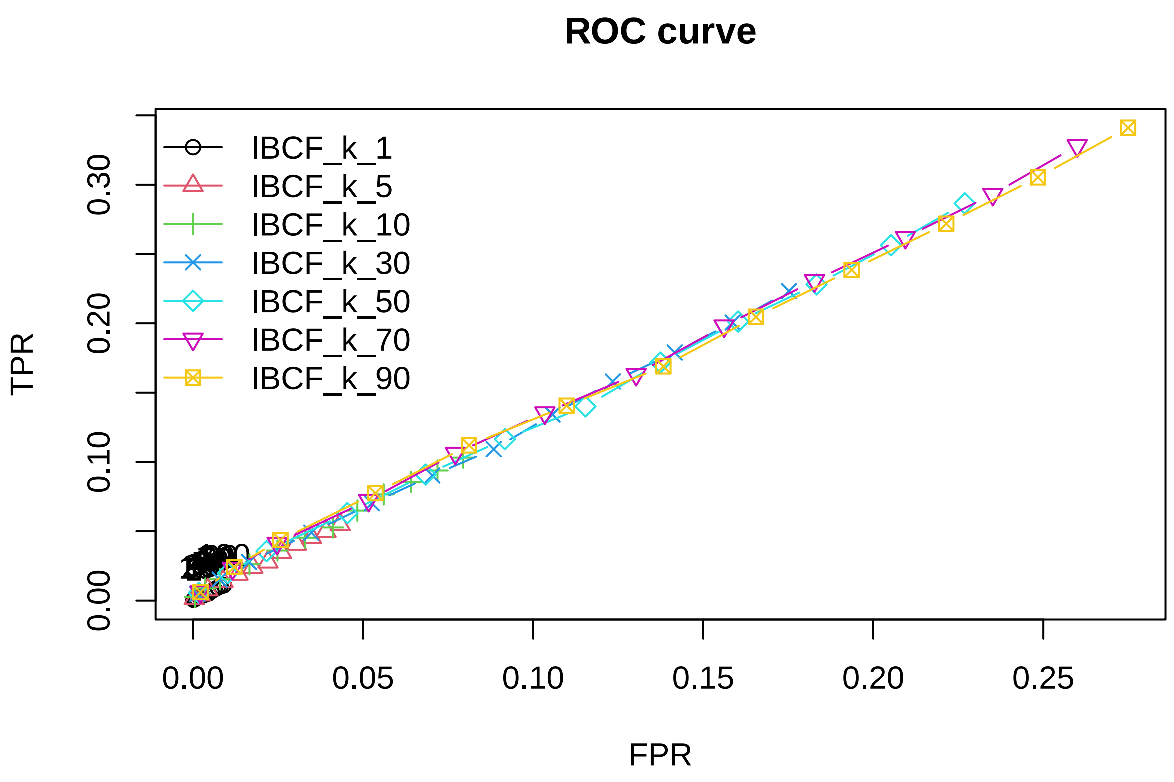
```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.18sec/0.02sec]
##   2  [0.17sec/0.02sec]
##   3  [0.17sec/0.02sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.21sec/0.01sec]
##   2  [0.19sec/0.02sec]
##   3  [0.17sec/0.01sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.19sec/0.01sec]
##   2  [0.15sec/0.01sec]
##   3  [0.18sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.17sec/0.03sec]
##   2  [0.18sec/0.03sec]
##   3  [0.18sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.17sec/0.03sec]
##   2  [0.17sec/0.02sec]
##   3  [0.18sec/0.04sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.19sec/0.03sec]
##   2  [0.18sec/0.03sec]
##   3  [0.17sec/0.05sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.17sec/0.05sec]
##   2  [0.18sec/0.06sec]
##   3  [0.17sec/0.04sec]
```

4.4.1 Understanding ROC Curves:

ROC (Receiver Operating Characteristic) curves are typically used in classification tasks, but in recommender systems, they can be adapted to evaluate the ability of a model to distinguish between relevant (liked) and irrelevant (not liked) items. On the x-axis is the False Positive Rate (FPR), and on the y-axis is the True Positive Rate (TPR). A model that performs random guessing will lie on the diagonal line (AUC = 0.5), while a good model will curve toward the top-left (AUC approaching 1).

In recommenderlab, ROC is computed by marking items with ratings above a "goodRating" threshold (e.g., 3) as relevant, then assessing how well the top-N recommendations retrieve these items across the test set. This gives us a way to compare performance across models consistently, even when recommendations are not explicitly rated.

```r
plot(list_results, annotate = 1,legend = "topleft")
  title("ROC curve")
```



5 Future Recommendation

To further improve this project, some potential future work involves the construction of hybrid recommender systems, trying matrix factorization models such as SVD or ALS, and incorporating deep learning methods. Also, adding time-based preferences and explainable recommendations can enhance personalization and users' trust.

5.1 Shortcomings of Current Approaches:

Both UBCF and IBCF are memory-based collaborative filtering techniques. They depend heavily on existing user ratings and struggle with:

Cold Start: New users or new items have no history, so no similarity can be computed.

Data Sparsity: The user-item matrix is extremely sparse, leading to less reliable similarity calculations.

Scalability: As the number of users or items grows, computing similarities becomes costly.

Why Matrix Factorization Models May Be Better: Techniques like SVD (Singular Value Decomposition) or ALS (Alternating Least Squares) can overcome sparsity by reducing the user-item matrix into lower-dimensional latent factors. These models:

- **Generalize better to unseen data.**
- **Can provide meaningful structure even when data is sparse.**
- **Are often more scalable due to efficient matrix operations.**

Deep Learning Models, such as neural collaborative filtering or autoencoders, can capture non-linear patterns and user-item interactions more flexibly, and may incorporate content data (text, genres) or temporal behaviors for richer personalization.

6 Conclusion

Here, we compared and contrasted two significant recommender system techniques: Item-Based Collaborative Filtering (IBCF) and User-Based Collaborative Filtering (UBCF) on the MovieLense dataset. By performing cross-validation and evaluating the performance using ROC curves, we concluded that IBCF with Jaccard similarity outperformed other models consistently for recommendation accuracy.

While both methods were promising, IBCF was superior in its ability to model item-level relationships and generate relevant recommendations. This demonstrates the usefulness of item similarity in sparse rating situations. Results also indicated limited performance gain through parameter tuning for settings like the number of neighbors.

The findings provide a good foundation for future enhancement by applying hybrid, matrix factorization, or deep learning models for further enhancement in recommendation quality and personalization.