Developing your own Movie Recommender

Dataset

To create our recommender, we use the data from movielens. These are film ratings from 0.5 (= bad) to 5 (= good) for over 9000 films from more than 600 users. The movieId is a unique mapping variable to merge the different datasets.

```
head(movie_data)
```

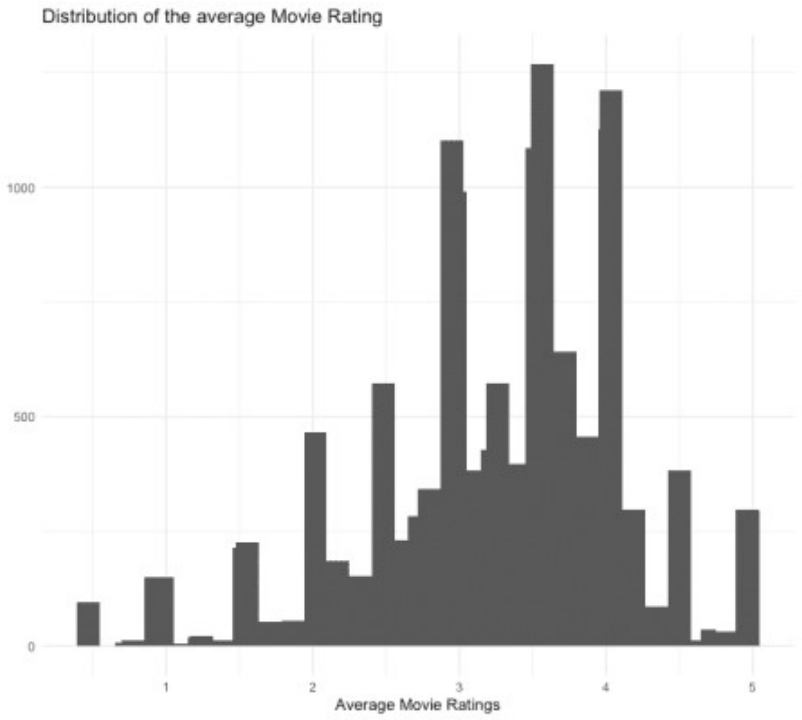
```
  movieId                               title
genres
1       1                   Toy Story (1995) Adventure|Animation|Children|
Comedy|Fantasy
2       2                     Jumanji (1995)
Adventure|Children|Fantasy
3       3            Grumpier Old Men (1995)
Comedy|Romance
4       4           Waiting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

```
head(ratings_data)
```

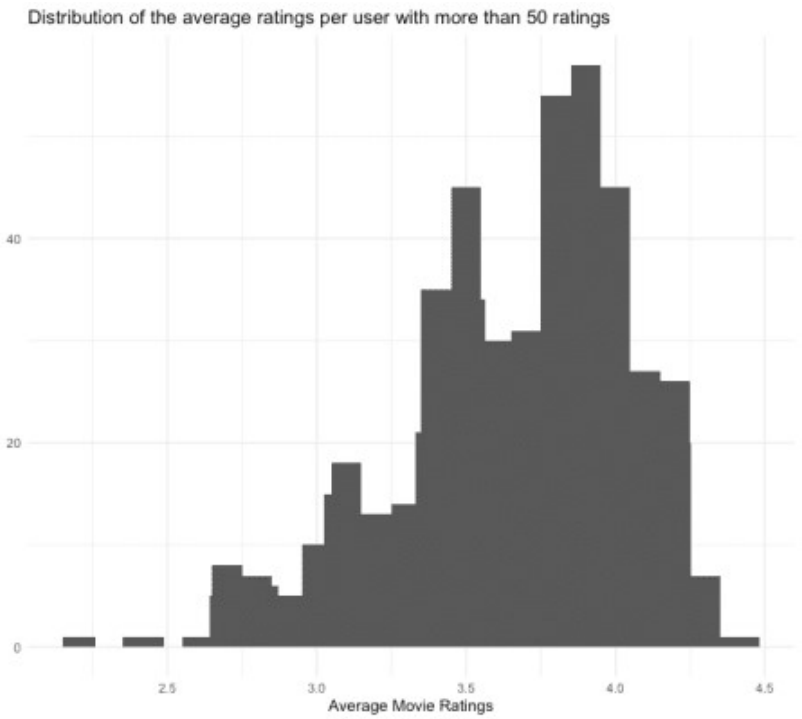
```
  userId movieId rating timestamp
1      1       1      4 964982703
2      1       3      4 964981247
3      1       6      4 964982224
4      1      47      5 964983815
5      1      50      5 964982931
6      1      70      3 964982400
```

To better understand the film ratings better, we display the number of different ranks and the average rating per film. We see that in most cases, there is no evaluation by a user. Furthermore, the average ratings contain a lot of „smooth“ ranks. These are movies that only have individual ratings, and therefore, the average score is determined by individual users.

```
# ranting_vector
0         0.5    1      1.5    2      2.5    3      3.5    4       4.5    5
5830804   1370   2811   1791   7551   5550   20047  13136  26818   8551   13211
```

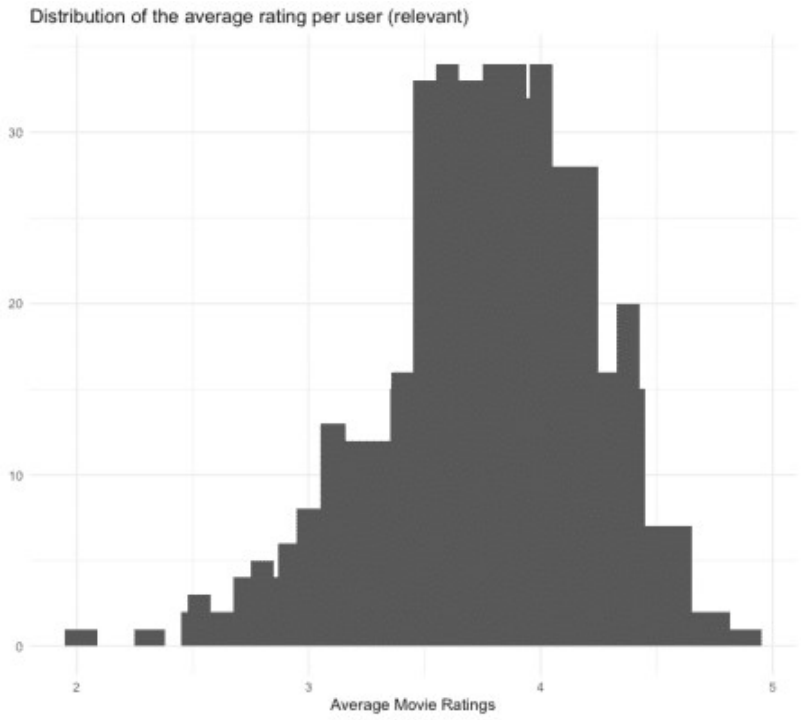


In order not to let individual users influence the movie ratings too much, the movies are reduced to those that have at least 50 ratings.




```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.208   3.444   3.748   3.665   3.944   4.429
```

Under the assumption that the ratings of users who regularly give their opinion are more precise, we also only consider users who have given at least 50 ratings. For the films filtered above, we receive the following average ratings per user:



You can see that the distribution of the average ratings is left-skewed, which means that many users tend to give rather good ratings. To compensate for this skewness, we normalize the data.

```
ratings_movies_norm <- normalize(ratings_movies)
```

Model Training and Evaluation

To train our recommender and subsequently evaluate it, we carry out a 10-fold cross-validation. Also, we train both an IBCF and a UBCF recommender, which in turn calculate the similarity measure via cosine similarity and Pearson correlation. A random recommendation is used as a benchmark. To evaluate how many recommendations can be given, different numbers are tested via the vector `n_recommendations`.

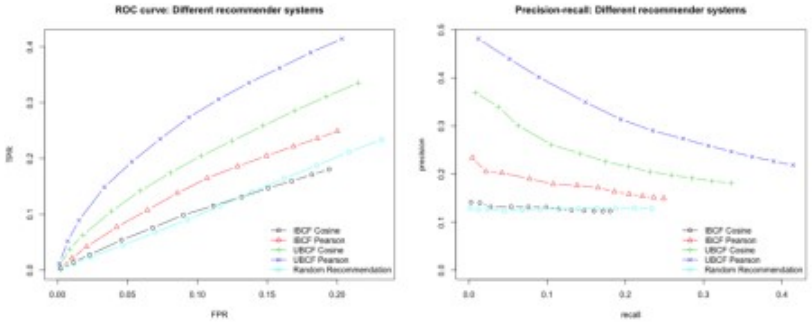
```
eval_sets <- evaluationScheme(data = ratings_movies_norm,
                              method = "cross-validation",
                              k = 10,
                              given = 5,
                              goodRating = 0)

models_to_evaluate <- list(
  `IBCF Cosinus` = list(name = "IBCF",
                        param = list(method = "cosine")),
  `IBCF Pearson` = list(name = "IBCF",
                        param = list(method = "pearson")),
  `UBCF Cosinus` = list(name = "UBCF",
                        param = list(method = "cosine")),
  `UBCF Pearson` = list(name = "UBCF",
                        param = list(method = "pearson")),
  `Zufälliger Vorschlag` = 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)
```

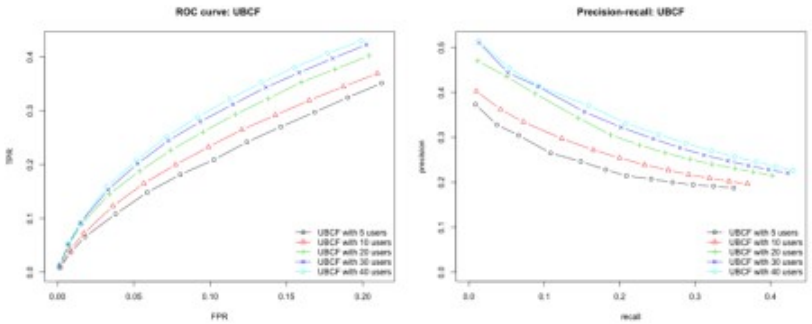
We then have the results displayed graphically for analysis.



We see that the best performing model is built by using UBCF and the Pearson correlation as a similarity measure. The model consistently achieves the highest true positive rate for the various false-positive rates and thus delivers the most relevant recommendations. Furthermore, we want to maximize the recall, which is also guaranteed at every level by the **UBCF Pearson** model. Since the n most similar users (parameter `nn`) are used to calculate the recommendations, we will examine the results of the model for different numbers of users.

```
vector_nn <- c(5, 10, 20, 30, 40)

models_to_evaluate <- lapply(vector_nn, function(nn){
  list(name = "UBCF",
       param = list(method = "pearson", nn = vector_nn))
})
names(models_to_evaluate) <- paste0("UBCF mit ", vector_nn, "Nutzern")
list_results <- evaluate(x = eval_sets,
                         method = models_to_evaluate,
                         n = n_recommendations)
```



Conclusion

Our user based collaborative filtering model with the Pearson correlation as a similarity measure and 40 users as a recommendation delivers the best results. To test the model by yourself and get movie suggestions for your own flavor, I created a small Shiny App.

However, there is no guarantee that the suggested movies really meet the individual taste. Not only is the underlying data set relatively small and can still be distorted by user ratings, but the tech giants also use other data such as age, gender, user behavior, etc. for their models.