Item-Based Collaborative Filtering Movie Recommendation System

Azra Park ([redacted])

05/04/2022

Introduction

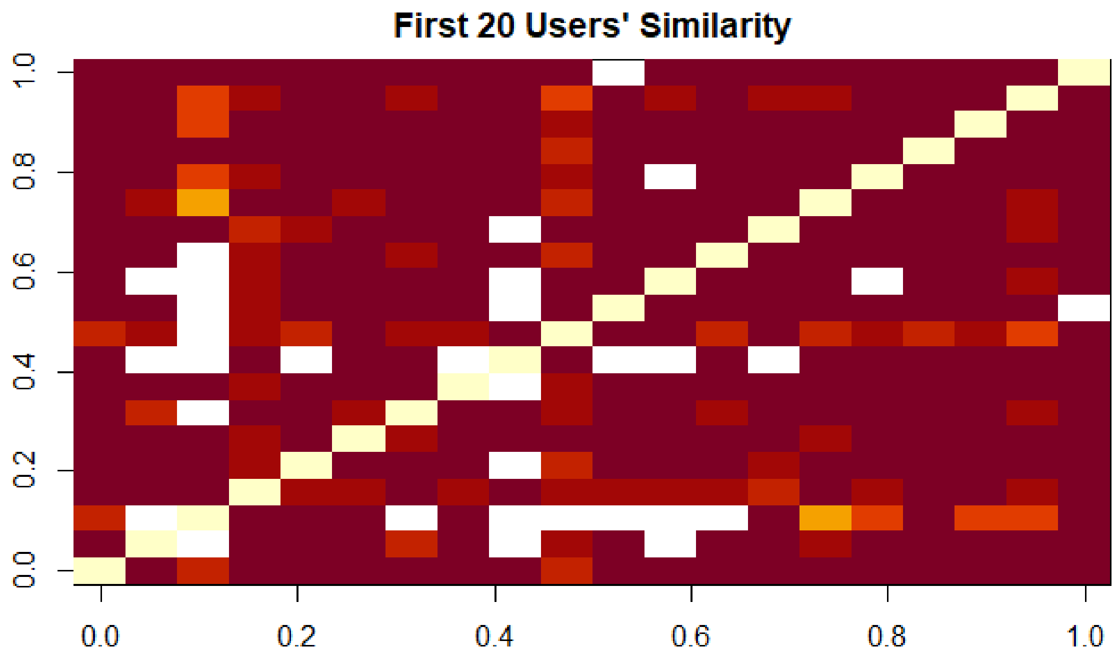
Recommendation systems are ML-based models that determine how similar users and products are to other things individual users and users like-them like, with the purpose of connecting them with recommendations among the plethora of selection in their given context. ("Machine Learning Project - Data Science Movie Recommendation System Project in r" 2019) In our given context, we will be looking at the dataset MovieLens in making a movie recommendation system, utilizing user ratings of various movies to then recommend the top movie recommendations for each user. To accomplish this, we will use item-based collaborative filtering in which given rating data by many users for items, the collaborative filtering algorithm finds similarity in the items based on the users' ratings for the purposes of predicting missing ratings and creating a top-N recommendation list for a given user. (Sharma 2018) Item-based collaborative filtering is a very popular method in recommendation systems first invented and implemented by Amazon in 1998. (Gupta 2020) The collaborative filtering algorithm we'll be using utilizes kNN regression techniques using cosine similarity to make these recommendations.

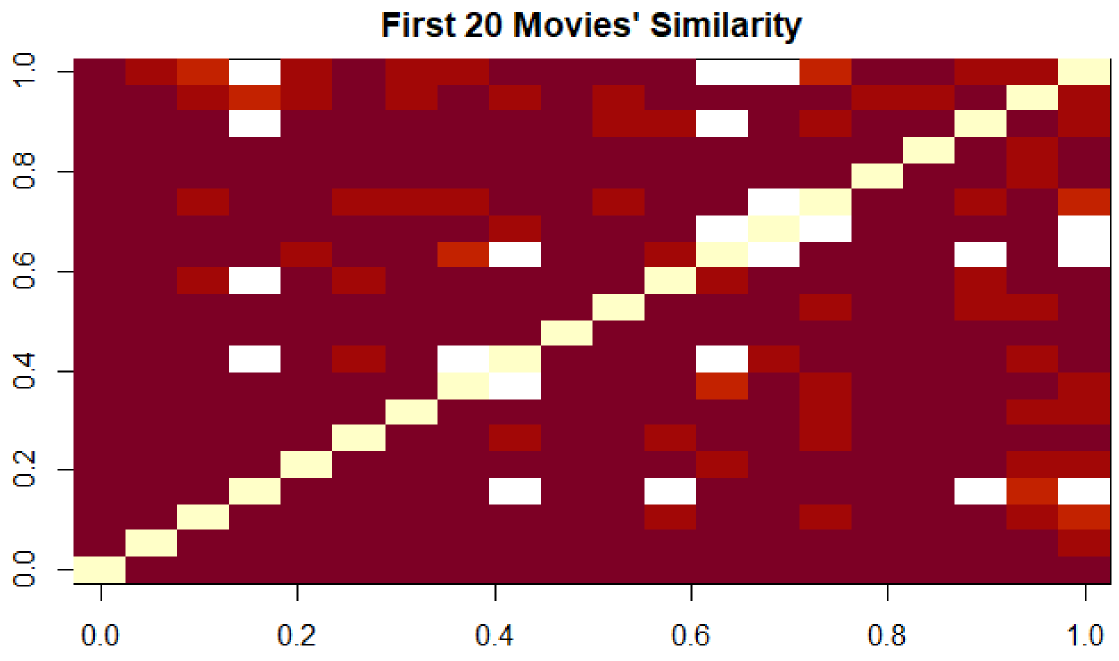
In this project, we will provide a description of the data, the methodology behind the item-based collaborative filtering, how the recommendation model is applied, and how we evaluate it. In the process we will build a movie recommendation system that will output a top 10 recommendation list for any given user.

Methodology (Models and Methods)

The application of recommendation systems is heavily integrated into the function of many services we interact with today, be it Netflix with movies, Spotify with music, LinkedIn with people, or Amazon with products. Much of what you are viewing on these services is decided based on a recommendation system based on your activity and the activity of other users like you (Hahsler 2021). This is to say that recommendation systems often use real-time datasets produced by the company itself on users, items, descriptors, as well as any other variables that may have an effect on your decisions when using the service. As for recommendation system projects, popular datasets include MovieLens, the dataset we will be using for movie recommendation systems, Jester5k, that makes for a joke recommendation system, the Netflix Prize dataset, that was used by Netflix to increase their recommendation systems' accuracy by competition, and many more.

As a beginning to my project, I apply some exploratory analysis of the ratings and movies datasets. As a first step, I must create a sparse rating matrix (a matrix that has the majority of the elements equal to zero) for the purposes of later applying the collaborative filtering technique to fill in the matrix with predicted ratings from those given. From there, I create two heatmaps examining the cosine similarities between the first 20 users and then the first 20 movies in our newly created sparse rating matrix. This provides insight into how the collaborative filtering technique perceives similarities between different users and items.





More specifically, in item-based collaborative filtering, cosine-based similarity items are represented in a user vector-space. The angle between two of these vectors are computed, the cosine of the angle being the similarity between the items. Cosine-based similarity has one drawback in that it does not account for individual user's rating biases as the calculations are performed over columns, with each column representing a different user. (Najafi and Salam 2016)

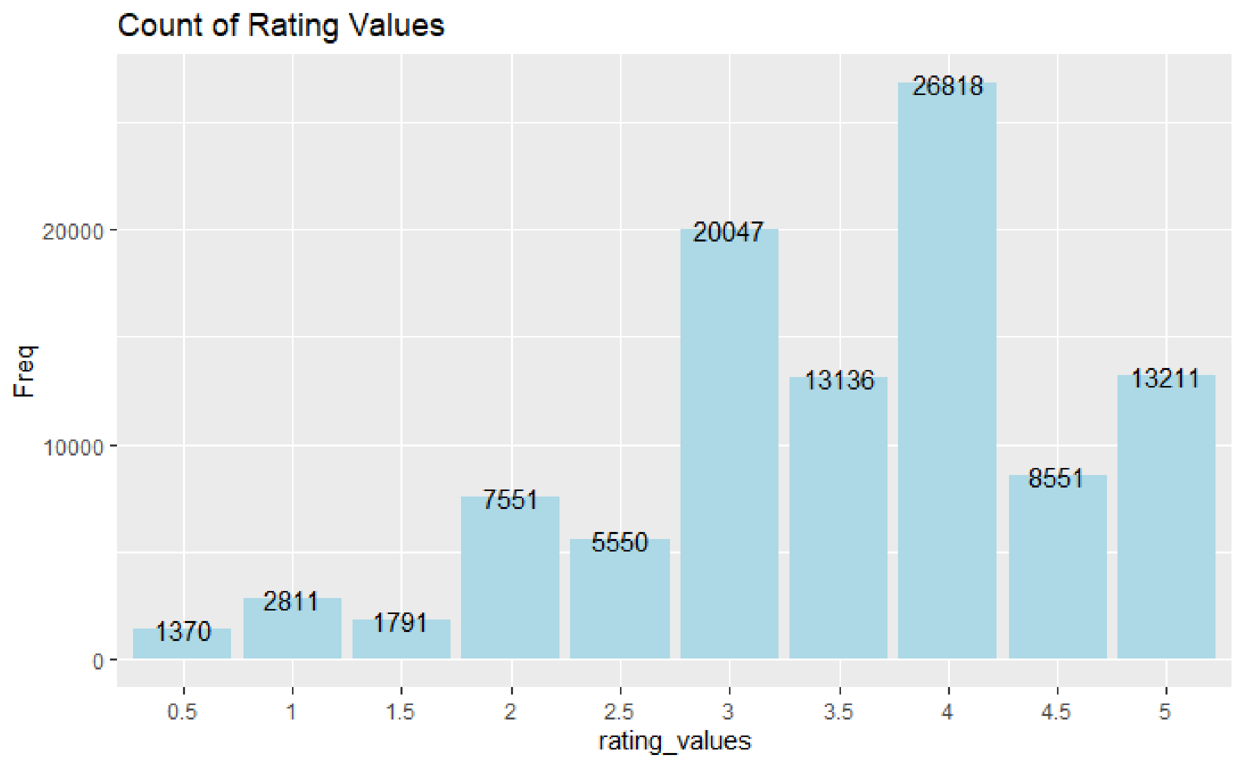
$$simil(x,y) = cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{||\vec{x}|| \times ||\vec{y}||} = \frac{\sum_{i\epsilon I_{x,y}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i\epsilon I_x} r_{x,i}^2}\sqrt{\sum_{i\epsilon I_y} r_{y,i}^2}}$$

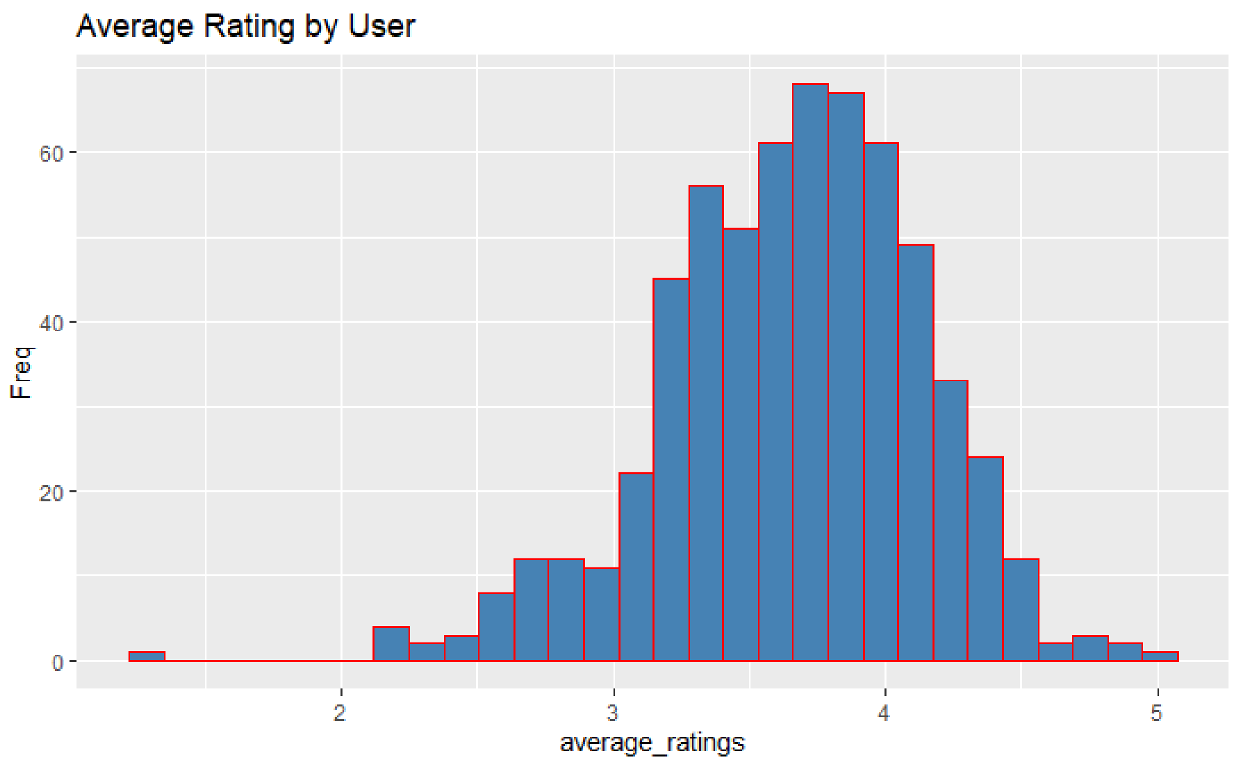
After calculating the cosine similarity, each empty element in the matrix is then given a predicted rating for any user-item pair by way of weighted sum:

$$P_{u,i} = \frac{\sum_{\text{all similar items},N}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items},N}(|s_{i,N}|)}$$

From all the items similar to our target item, we pick the items which an individual user has rated, then weight the user's rating for each of these items by the similarity between the target item and each of the similar items. Finally, the prediction is scaled by the sum of similarities to get the predicted ratings for each missing item, this done for each user filling in the initial sparse matrix. (Sarwar et al. 2001; "Algorithsm :: Item-Based Collaborative Filtering," n.d.)

I then look at the unique ratings users are able to apply to movies, which range from 0.5 to 5, in 0.5 increments. Although 0 is included in the count, 0 is equivalent to no rating and is thus not included in the bar plot. A bar plot is used as a visual aid for comprehending each unique ratings' total count found in the sparse rating matrix. This is then further supplemented by a histogram of each individual user's average rating, as a means of examining bias in the data.





Now beyond exploratory data analysis, I begin optimizing for k, the number of nearest neighbours to each user in relation to their cosine similarities, with consideration to evaluation metrics in preparation for use in the recommendation system and its performance. In order to retain strong predictions, I first filter out users and movies that do not have a sufficient amount of data in order to ensure reliable predictions. Although this approach is not wholly practical, this does mitigate the cold start problem due to data sparsity often encountered in recommendation systems built on collaborative filtering. The cold start problem being that with users and items with little to no data points, it is difficult to accurately provide recommendations as collaborative filtering methods require a history of user preferences. My evaluation starts with creating an evaluation scheme that splits movie_ratings into a training set (80%) and a test set (20%). For the test set, all-but 4 items will be given to the recommendation algorithm; the algorithm sees all but 4 randomly selected withheld ratings for each test user and the algorithm is evaluated by how well it is able to predict the withheld items. This was chosen on the basis of optimal accuracy in consideration to the evaluation metrics. With using a realRatingMatrix, we are required to supply a goodRating threshold for which ratings are considered good for evaluation for each user. I decided to set this to 4.5, as to only capture the top recommendations for each user as a higher predicted rating makes for a "better" recommendation, as well as increasing the accuracy of the top-N list. Upon trying cross-validation and bootstrapping, I found them to be much slower and not as accurate as using the split method. Recommendation systems must be extremely time efficient due to the applications recommendation systems are used, such is the case with an ever updating log of user preferences and the requirement of having recommendations available while many users browse selection, this is not to say this performance cannot be achieved with cross-validation and bootstrapping, this is just my case in the implementation of this recommendation system.

As a next step, I create a function to optimize for k (# of neighbours) that is applied in the process of when the recommendation model groups the k-most similar items based on cosine similarity for identifying user specific recommendations. In this function, we optimize k in consideration to the True Positive Rate (TPR) and False Positive Rate (FPR) of the top-N list of movies, as that is the end objective after all, accurate movie recommendations. TPR measures the percentage of actual positives that are accurately identified, while FPR measures the percentage of actual negatives incorrectly categorized as positive. A successful TPR value is still fairly low as recommendations are still difficult to predict due to the spectrum of user tastes and preferences, oftentimes a good value for TPR is as low as 0.08, the higher the TPR the better. However, this is not uncommon, we can simply consider how many items on Netflix, Amazon, or any other service utilizing recommendation systems we actually consume; most items that are recommended to us we do not consume. A good value for FPR is approximately 0.015, as this is to say there was a bad recommendation the user would not like in the top-N recommendation list, the lower the FPR the better.

$$TPR = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

$$FPR = \frac{\text{False Positives}}{\text{True Negatives + False Positives}}$$

Upon finding the k value with the (relative) highest TPR and lowest FPR, we then use this k value in finding the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) for our predicted ratings for which our recommendations are based on. RMSE and MAE are used for evaluating the predicted ratings against the actual user ratings in order to measure how accurate the filtering technique performs in filling-in the missing data (Doshi 2018). MAE computes the average of all the absolute value differences between the true and the predicted rating, while RMSE computes the square root of the mean value of all the squared differences between the true and predicted ratings (Najafi and Salam 2016). In both cases, the lower the error, the more accurately the recommendation engine predicts user ratings.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y_i} - y_i)^2}{n}}$$

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y_i}|$$

An example of a successful test RMSE comes from the famous Netflix Prize in which Netflix had a grand prize of US$1,000,000, for any participating team that could improve the recommendation algorithm by 10%, or in other words that could achieve an RMSE of 0.8572 on the test set (Bennett and Lanning 2007). Though, in general, a successful test RMSE is approximately 0.9 with room to be higher than that, while a successful MAE is approximately 0.8, again with room beyond that. In order to make these calculations, we first use the predict() function to generate predictions for the known portion of the test data, then use the calcPredictAccuracy() function to calculate the error between the predictions and the unknown portions of the test data, with test RMSE, MSE, and MAE values being returned. (Topor 2017) Another key point to mention is that for rating matrices it is important to normalize the ratings in order to remove the user rating bias. However, this is done by default in the function Recommender() that creates our model. Finally, I construct the function that predicts and returns the top 10 movie recommendations for any given user using the recommendation model we created before with the known test data.

Data Description

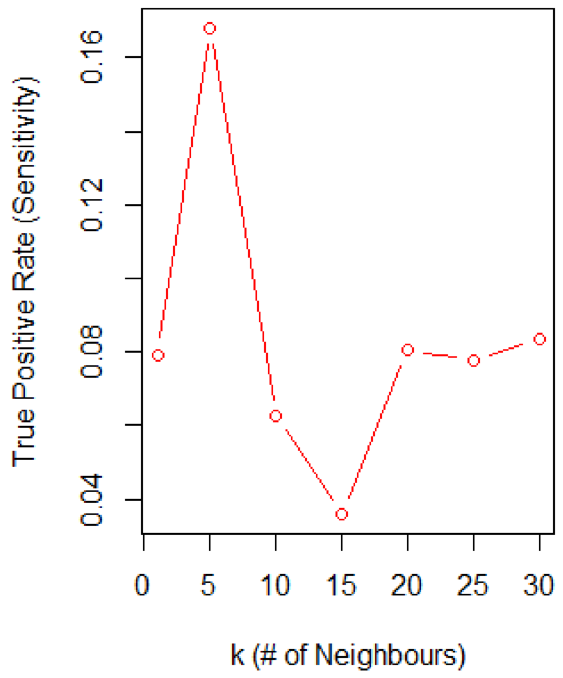
I will be using a popular dataset used for Movie Recommendation Systems projects, MovieLens. Due to computing power limitations, a smaller subset of the larger dataset was used. This dataset contains 100,836 ratings applied to 9,742 movies by 610 users. The movies span across 19 genres and have 3,683 tags (descriptors) provided by the users in reviews, though these variables were not considered in the recommendation system as they were not applicable to the method of item-based collaborative filtering used. The MovieLens dataset is taken from GroupLens, a research lab in University of Minnesota's Department of Computer Science and Engineering specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems (GroupLens 2019). The data is collected via the MovieLens website (http://movielens.org) that hosts hundreds of thousands of registered users in helping them find personalized movie recommendations.

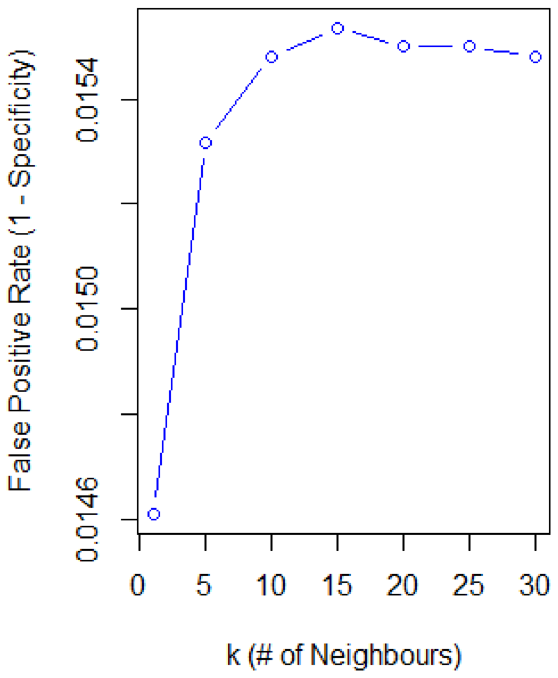
Model Fitting

During the process of fitting the model, there were a few procedures and decisions made in how to handle the data using statistical practices. For instance, one of the first requirements for processing the data was creating a sparse matrix that contained all the users and their ratings for all the movies which would be later filled in with predicted ratings from those given in the datasets. One of the most impactful changes in the data used in the recommendation system was the creation of the movie_ratings matrix that took the initial sparse rating matrix and filtered out any users and movies that did not meet the required amount of elements (set to 40) for the purposes of retaining reliable predictions. In creating an evaluation scheme for the recommendation system, I decided to choose the split method with an 80/20 split due to the required level of diversity between the train and test data in order to provide varied recommendations curated for the niche preferences of individuals; effect on evaluation metrics was also considered. I found using split to be the most accurate method, but also the fastest. This is important, as mentioned before recommendation systems must be extremely efficient in time and memory usage due to the scalability requirements of having recommendations readily available for millions of users. (Linden, Smith, and York 2003) In the evaluation scheme, I also set the goodRating parameter to 4.5 for the model to be particularly discriminate in what makes a movie a personal recommendation rather than just a good movie. Then, set the given parameter to All-but-4 in order to maximize our evaluation metrics while still performing a comprehensive testing method in which the algorithm is evaluated on it's ability to predict the 4 withheld items. (Hahsler 2021) I then built the IBCF_tune function to find the accuracy of the Top 10 recommendation list, to be implemented into topN_performance to find the optimal k that provided the highest TPR and lowest FPR for the most reliable recommendations. From this, I found that 5 was the optimal k to which I would use in the recommendation model and in finding the predicted ratings evaluation metrics, RMSE and MAE. This is important as the recommendation system is founded on the predicted ratings from collaborative filtering technique using cosine similarity that these metrics evaluate. I also set the Recommender() function to normalize the data by centering to remove users' rating biases.

Results

I found the results of the recommendation system to be quite accurate with respect to the strong evaluation metrics, as well as the recommendations outputted for each user. I mention the recommendations themselves, as given my knowledge of movies, I was able to detect patterns in user preferences between movies and found cases in which movie series would have multiple movie entries, such as sequels, appear in the top 10 list which would be expected given the close cosine similarities. Although that is an example of mere heuristics, my evaluation metrics proved to be competitive with many of the projects and cases I had reviewed in researching this project. A good example of this is the Netflix Prize test RMSE of 0.8572, while I was able to achieve a test RMSE of 0.8652 and an MAE of 0.5980 in regards to my predicted ratings, where I found most reliable models using various datasets, including the one I had used, to produce approximately an MAE of 0.75. I also was able to achieve a strong TPR of 16.8% and FPR of 1.5% to which I found other examples to have a TPR closer to 9% and an FPR of 1.6% in regards to the accuracy of the outputted top 10 recommendations for each user (ie: my model has a probability of 16.8% to place an appealing movie in the top-10). (Cremonesi, Koren, and Turrin 2010; Sarwar et al. 2001)





I don't believe there is anything I would change to improve results, however I do sense that the parameters chosen may be too restrictive. For example, requiring users and movies to have at least 40 elements to be applied in the recommendation system or the setting of a high goodRating threshold parameter in the evaluation scheme, though this is not to say these measures did not have a beneficial purpose as they increased the accuracy of the model. I also felt the validation process was limited, but appropriate for the scope of the project, though I found each component of the recommendation system to have the effect I would expect especially seen in the choice of parameters. This is to say confidence in results are fairly low, but that is often the case with recommendation systems given the requirements of scalability in their applications with a trade-off between the level of prediction and speed, as well as the dubious task required of these algorithms in curating to the very subjective nature of user preferences, hence the expectation of a low TPR in our recommendations (Cremonesi, Koren, and Turrin 2010; Couch 2020).

Conclusion

Overall, I am quite pleased with the results of my project, and more importantly I learned a lot. I find that my recommendation system not only works, but is also quite competitive in performance to others I had seen in the process. I found there were many limitations to this project, often simply due to the many special cases to consider much beyond the scope of the project. Some limitations include that in industry these algorithms are performed on a consistently updated dataset, often considering each individual session, as well as all past history which are implemented via data pipelines to automate these processes. Another limitation being that by requiring to have at least 40 data points for each movie and user as a means of retaining the quality of recommendations, we excluded a large selection of movies and users in the process. Though, it can be argued that there is something to be said for this approach as not only does it address the cold start problem, but there is a strong likelihood that movies with at least 40 watches would make for better recommendations anyway, given they are more popular than those excluded. With this in mind, one of the recommendation models often used functions on the basis of item popularity. In the same manner of movies and users being excluded due to requiring a certain amount of data, due to computing power limitations we were required to use a smaller subset of the larger MovieLens dataset, which restricts the movies included in the recommendation system, as well as reducing the potential accuracy of predictions due to having less data points. It is also important to consider that there are many approaches to creating a recommendation system, and it did not make sense to explore each caveat due to the time constraints and scope of the project. Be it simply using different recommendation models or using a larger set of variables for making recommendations such as with tags/descriptors, timestamps, etc. In closing, this project taught me the few key requirements that recommendation systems rely on, making for easy future reference and use. I believe it is an important piece of data science to learn, especially as these algorithms play a larger role in how society navigates life and makes decisions.

Appendix: R Code

```
library(recommenderlab)
library(tidyverse)
library(reshape2)
library(data.table)
```

Exploratory Analysis

```
set.seed(3)
movies_df <- read.csv('movies.csv')
ratings_df <- read.csv('ratings.csv')

head(movies_df)
```

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

```
head(ratings_df)
```

```
##   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
```

```
#Summary statistics
summary(movies_df)
```

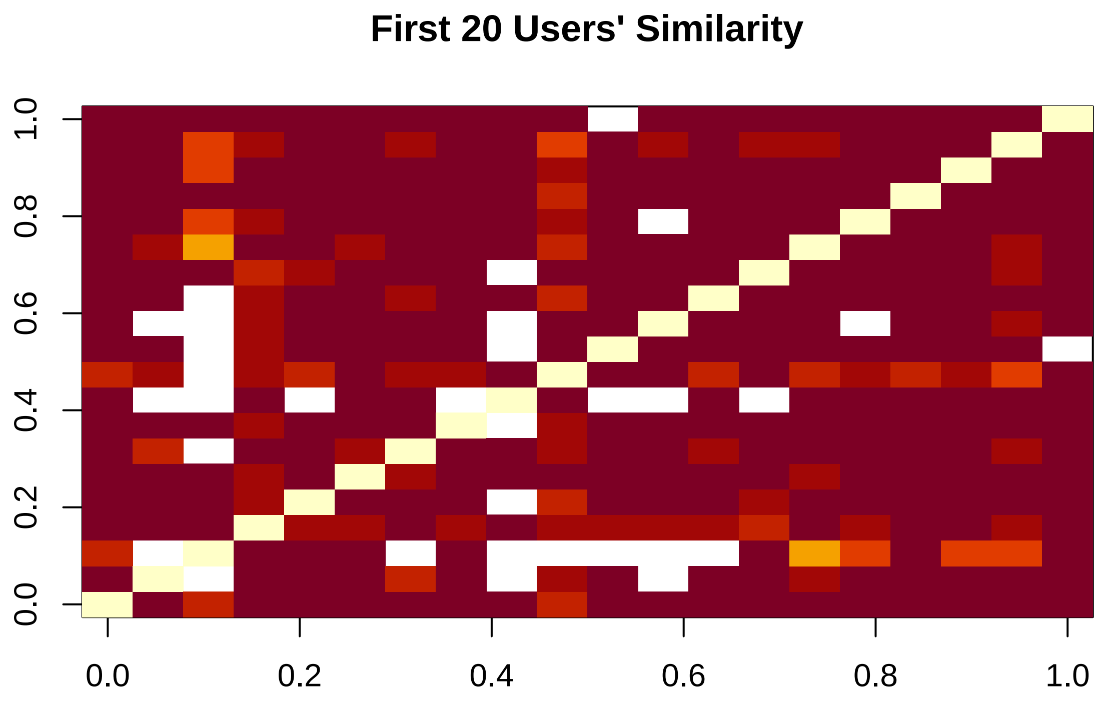
```
##     movieId          title              genres
##  Min.   :     1   Length:9742        Length:9742
##  1st Qu.:  3248   Class :character   Class :character
##  Median :  7300   Mode  :character   Mode  :character
##  Mean   : 42200
##  3rd Qu.: 76232
##  Max.   :193609
```

```
summary(ratings_df)
```

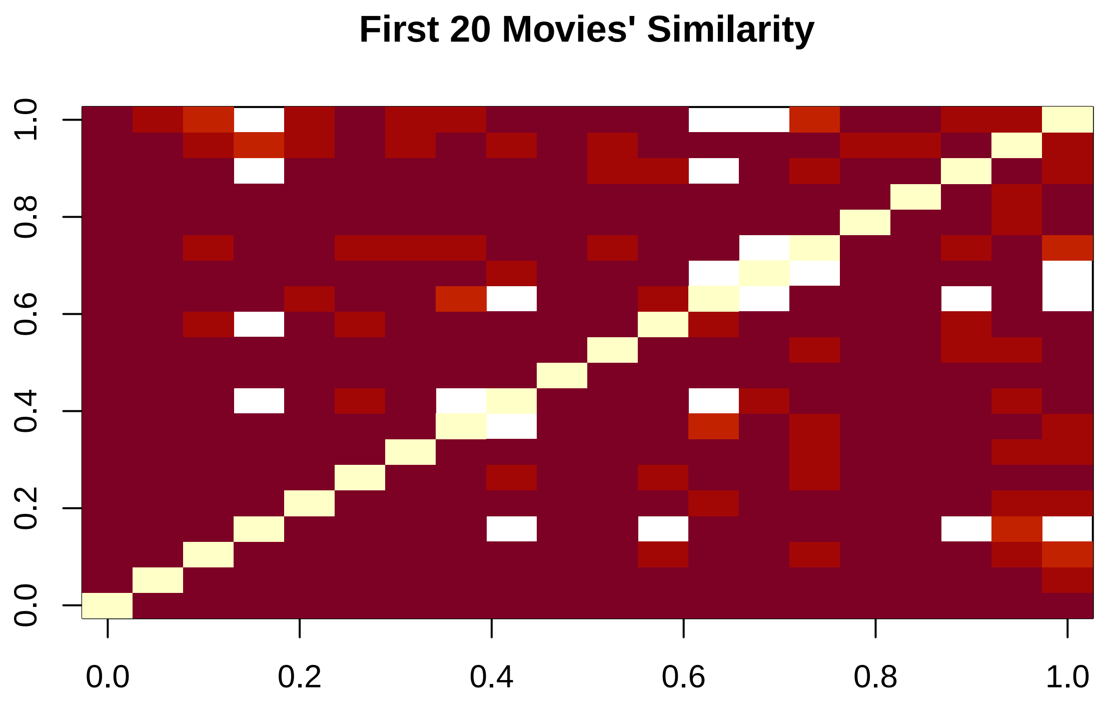
```
##      userId          movieId            rating        timestamp
##  Min.   :  1.0   Min.   :     1   Min.   :0.500   Min.   :8.281e+08
##  1st Qu.:177.0   1st Qu.:  1199   1st Qu.:3.000   1st Qu.:1.019e+09
##  Median :325.0   Median :  2991   Median :3.500   Median :1.186e+09
##  Mean   :326.1   Mean   : 19435   Mean   :3.502   Mean   :1.206e+09
##  3rd Qu.:477.0   3rd Qu.:  8122   3rd Qu.:4.000   3rd Qu.:1.436e+09
##  Max.   :610.0   Max.   :193609   Max.   :5.000   Max.   :1.538e+09
```

```
#Creates sparse matrix for recommendation
rating_matrix <- reshape2::dcast(ratings_df, userId~movieId, value.var = 'rating', na.rm = FALSE)
rating_matrix <- as.matrix(rating_matrix)
rating_matrix <- as(rating_matrix[,2:9725], 'realRatingMatrix')

#User's Similarity
user_similarity_matrix <- similarity(rating_matrix[1:20, ], method = 'cosine', which = 'users')
image(as.matrix(user_similarity_matrix), main = 'First 20 Users\' Similarity')
```



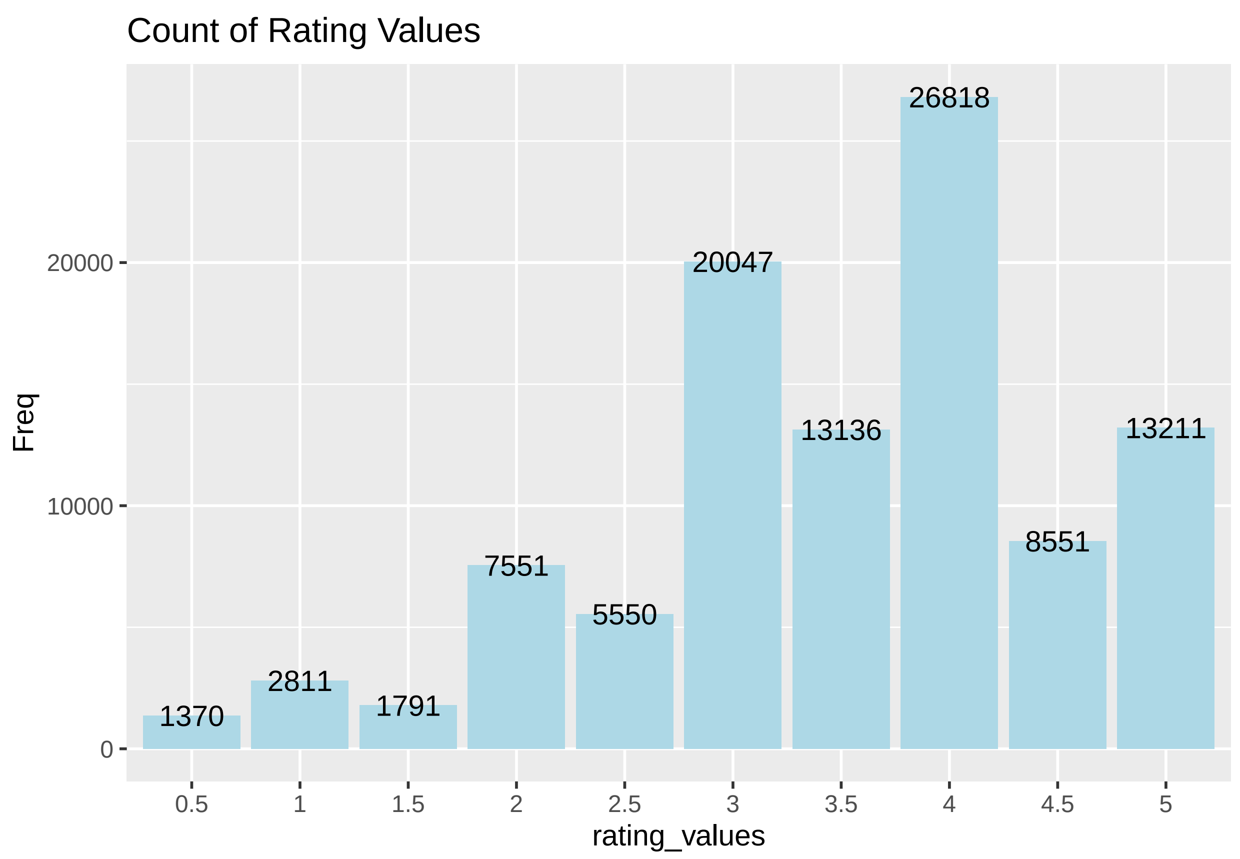
```
#Movie's Similarity
movie_similarity_matrix <- similarity(rating_matrix[,1:20], method = 'cosine', which = 'items')
image(as.matrix(movie_similarity_matrix), main = 'First 20 Movies\' Similarity')
```



```
#Rating values
rating_values <- as.vector(rating_matrix@data)
rating_table <- table(rating_values)
rating_table
```

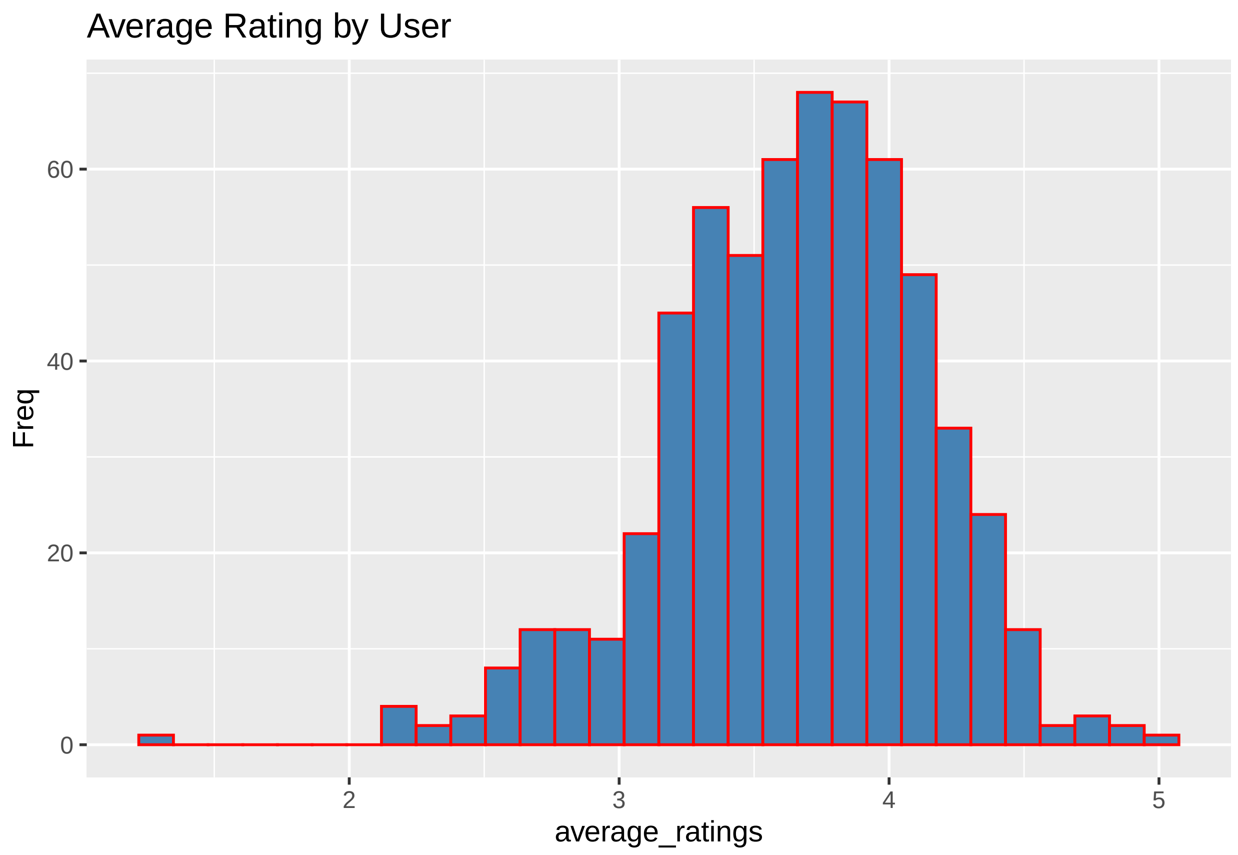
```
## rating_values
##       0     0.5       1     1.5       2     2.5       3     3.5       4     4.5
## 5830804    1370    2811    1791    7551    5550   20047   13136   26818    8551
##       5
##   13211
```

```
ggplot(as.data.frame(rating_table[2:11]),aes(rating_values,Freq)) +
geom_bar(stat = "identity", fill = 'light blue') +
ggtitle('Count of Rating Values') +
geom_text(aes(label = signif(Freq)))
```



```
average_ratings <- rowMeans(rating_matrix)
qplot(average_ratings, fill = I('steelblue'), col = I('red')) +
  ggtitle('Average Rating by User')+
    ylab("Freq")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
#We see a positive rating bias that will be accounted for in the Recommender model

#Filter Useful Data, using central limit theorem
movie_ratings <- rating_matrix[rowCounts(rating_matrix) >= 40,
                               colCounts(rating_matrix) >= 40]
```

[“Machine Learning Project - Data Science Movie Recommendation System Project in r” (2019);hahsler_2021_recommenderlab]

Collaborative Filtering System

```
set.seed(3)
recommendation_system <- recommenderRegistry$get_entries(dataType = 'realRatingMatrix')
#Default parameters
recommendation_system$IBCF_realRatingMatrix$parameters
```

```
## $k
## [1] 30
##
## $method
## [1] "Cosine"
##
## $normalize
## [1] "center"
##
## $normalize_sim_matrix
## [1] FALSE
##
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE
```

```
model_evaluation <- evaluationScheme(movie_ratings, method="split",
                                     train=0.8, given=-4, goodRating=4.5)
```

(Hahsler 2021)

Optimization and Evaluation

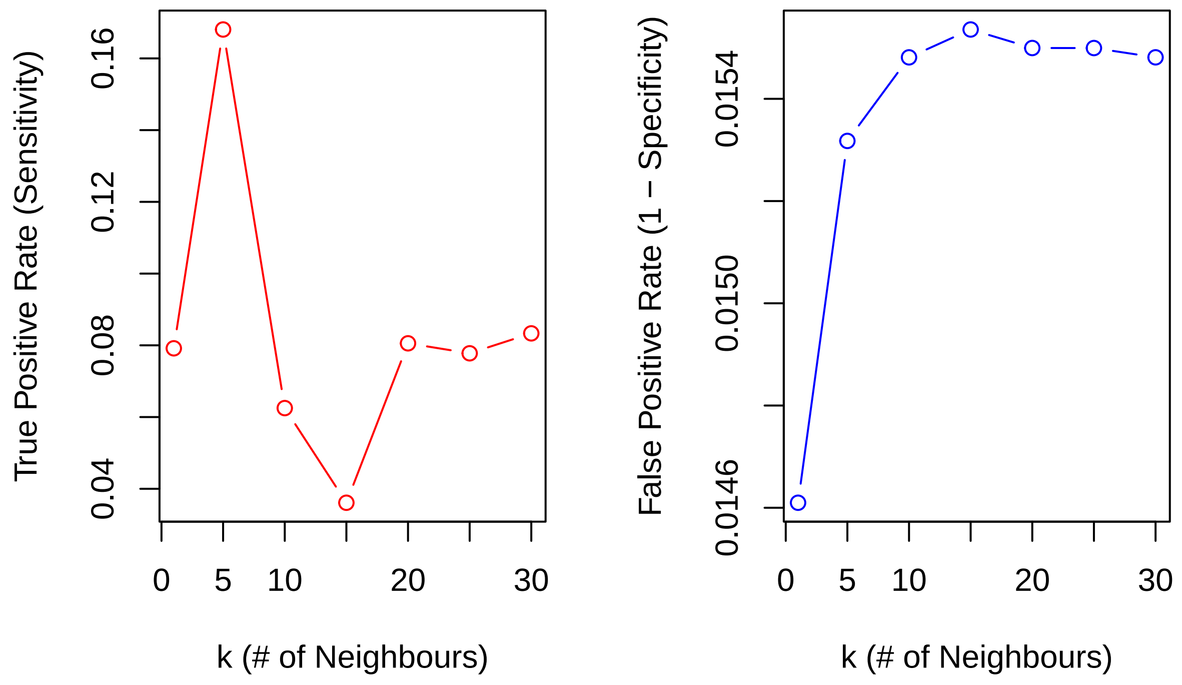
```
#Find optimal k
IBCF_tune <- function(evalScheme, parameters){
  IBCF_topN <- evaluate(model_evaluation, method = 'IBCF',
                        type = 'topNList', n = 10, param = list(k = parameters))
  IBCF_topN %>%
    avg() %>%
    as_tibble() %>%
    mutate(param = parameters, model = 'IBCF') %>%
    return()
}
tune_grid <- tibble(parameters = c(1, 5, 10, 15, 20, 25, 30))
topN_performance <- tune_grid %>%
           mutate(results = map(parameters, ~IBCF_tune(model_evaluation, .x))) %>%
           unnest(cols = c(results))
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.45sec/0.08sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.48sec/0.07sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.44sec/0.01sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.42sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.43sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.43sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.42sec/0.03sec]
```

```
topN_performance <- as.data.frame(topN_performance)
topN_performance
```

```
##   parameters         TP       FP       FN       TN   N   precision     recall
## 1          1 0.11627907 9.360465 1.197674 631.3256 642 0.012126246 0.07916667
## 2          5 0.18604651 9.813953 1.127907 630.8721 642 0.018604651 0.16805556
## 3         10 0.08139535 9.918605 1.232558 630.7674 642 0.008139535 0.06250000
## 4         15 0.04651163 9.953488 1.267442 630.7326 642 0.004651163 0.03611111
## 5         20 0.06976744 9.930233 1.244186 630.7558 642 0.006976744 0.08055556
## 6         25 0.06976744 9.930233 1.244186 630.7558 642 0.006976744 0.07777778
## 7         30 0.08139535 9.918605 1.232558 630.7674 642 0.008139535 0.08333333
##          TPR        FPR  n param model
## 1 0.07916667 0.01460981 10     1  IBCF
## 2 0.16805556 0.01531762 10     5  IBCF
## 3 0.06250000 0.01548111 10    10  IBCF
## 4 0.03611111 0.01553561 10    15  IBCF
## 5 0.08055556 0.01549939 10    20  IBCF
## 6 0.07777778 0.01549936 10    25  IBCF
## 7 0.08333333 0.01548116 10    30  IBCF
```

```r
par(mfrow=c(1,2))
plot(topN_performance$parameters, topN_performance$TPR, type="b", col="red",
     xlab="k (# of Neighbours)", ylab="True Positive Rate (Sensitivity)")
abline()
plot(topN_performance$parameters, topN_performance$FPR, type = 'b', col="blue",
     xlab="k (# of Neighbours)", ylab="False Positive Rate (1 - Specificity)")
```



```r
#With optimal k chosen, we can view our RMSE and MAE for the predicted ratings
recommend_model <- Recommender(getData(model_evaluation, "train"), "IBCF",
                               param=list(normalize = 'center', method="Cosine", k = 5))

predictions <- predict(recommend_model, getData(model_evaluation, "known"), type="ratings")
error <- calcPredictionAccuracy(predictions, getData(model_evaluation, "unknown"))
error
```

```
##      RMSE       MSE       MAE
## 0.8652214 0.7486080 0.5980230
```

[Couch (2020); topor_2017_rpubs;hahsler_2021_recommenderlab]

Recommendation system

```r
set.seed(3)
top_10 <- function(user){
  predicted_recommendations <- predict(object = recommend_model,
                                       newdata = getData(model_evaluation, "known"),
                                       n = 10)
  user_recommended_titles <- predicted_recommendations@itemLabels[predicted_recommendations@items[[user
  user_recommended_titles_copy <- user_recommended_titles
  for (index in 1:10){
    user_recommended_titles_copy[index] <- as.character(subset(movies_df,
                                           movies_df$movieId == user_recommended_titles[index])$title)
  }
  user_recommended_titles_copy
}

#Top 10 recommendations for user 1
top_10(1)
```

```
##  [1] "When Harry Met Sally... (1989)"
##  [2] "Close Encounters of the Third Kind (1977)"
##  [3] "Naked Gun: From the Files of Police Squad!, The (1988)"
##  [4] "Beverly Hills Cop (1984)"
##  [5] "Pulp Fiction (1994)"
##  [6] "Kingpin (1996)"
##  [7] "2001: A Space Odyssey (1968)"
##  [8] "Raging Bull (1980)"
##  [9] "Chinatown (1974)"
## [10] "Raising Arizona (1987)"
```

References

"Algorithsm :: Item-Based Collaborative Filtering." n.d. cs.carleton.edu. https://cs.carleton.edu/cs_comps/0607/recommend/recommender/itembased.html.

Bennett, James, and Stan Lanning. 2007. "The Netflix Prize." https://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf.

Couch, Andrew. 2020. "TidyTuesday: Creating a Song Recommendation Engine in r." Youtube. https://www.youtube.com/watch?v=U4C6G_lHn04.

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin. 2010. "Performance of Recommender Algorithms on Top-n Recommendation Tasks." *Proceedings of the Fourth ACM Conference on Recommender Systems - RecSys '10*, September, 39–46. https://doi.org/10.1145/1864708.1864721.

Doshi, Neerja. 2018. "Recommendation Systems—Models and Evaluation." Medium; Towards Data Science. https://towardsdatascience.com/recommendation-systems-models-and-evaluation-84944a84fb8e.

GroupLens. 2019. "MovieLens." GroupLens. https://grouplens.org/datasets/movielens/.

Gupta, Rachit. 2020. "Item-to-Item Based Collaborative Filtering." GeeksforGeeks. https://www.geeksforgeeks.org/item-to-item-based-collaborative-filtering/.

Hahsler, Michael. 2021. "Recommenderlab: A Framework for Developing and Testing Recommendation Algorithms." https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf.

Linden, Greg, Brent Smith, and Jeremy York. 2003. "Amazon Recommendations." https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf.

"Machine Learning Project - Data Science Movie Recommendation System Project in r." 2019. DataFlair. https://data-flair.training/blogs/data-science-r-movie-recommendation/.

Najafi, Safir, and Ziad Salam. 2016. "Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems." https://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A927356&dswid=5732.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Reidl. 2001. "Item-Based Collaborative Filtering Recommendation Algorithms." *Proceedings of the Tenth International Conference on World Wide Web - WWW '01*, April, 285–95. https://doi.org/10.1145/371920.372071.

Sharma, Anupam. 2018. "RecommenderSystems." GitHub. https://github.com/AnupamMicrosoft/RecommenderSystems.

Topor, James. 2017. "RPubs - User-Based and Item-Based Collaborative Filtering." https://rpubs.com/jt_rpubs/285729.